Desarrollo de una canalización para el testeo automatizado en el ámbito del desarrollo de aplicaciones iOS

Titulación: Grado en Ingeniería Informática

Autor: Jorge Padrón Seisdedos

TUTORIZADO POR:

Francisco Alexis Quesada Arencibia

Junio de 2022

Índice

Índice Ilustraciones

Índice Tablas de Comandos

Índice Tablas

Resumen

Este proyecto surge a partir de la necesidad de mejorar la producción de software. Esta mejoría se centra en la automatización de los procesos de pruebas dentro de las aplicaciones producidas bajo el entorno iOS, pudiendo implementar también metodologías como *test driven development* (TDD), que mejorarían la calidad del producto.

Con esta idea se propone la creación de una canalización capaz de integrarse a distintos repositorios, automatizando la fase de pruebas de los proyectos de los distintos equipos de desarrollo, buscando un solo servicio que sea flexible, de fácil implementación y altamente personalizable según las necesidades del cliente, pudiendo ser reutilizable. Además, se propone la generación de informes sobre el desarrollo de las pruebas, añadiendo gran valor al proyecto.

Abstract

This project arises from the need of the market of improving the software development, this improvement is focused on the automatization of all the processes of testing within the development of application under iOS environment. Also giving the chance of implementing methodologies such as test-driven development which could improve the quality of the project.

Based on this, we came up with the idea of developing a pipeline that is able to adhere to different repositories, automating the phase of testing on the projects of various development teams. Using this approach, we could make a service which is flexible, highly customizable depending on the needs of the clients and reusable. Furthermore, we want to the collect all the results of the test and generate a report with them, adding a lot of value to this project.

1. Introducción

Durante el periodo de prácticas que el autor de esta memoria cursó en *Advance Digital Experts*, las metodologías ágiles como *SCRUM* y el desarrollo dirigido por pruebas fueron aplicadas, y seguirlas fue lo estándar. Sin embargo, no siempre estas metodologías se aplican correctamente. En muchos equipos de desarrollo, hay fases del desarrollo software que no se ejecutan o no se efectúan de la forma adecuada por diversos motivos.

Una de las fases más comunes del desarrollo de software es la prueba del código que conforma el programa. Gracias a esto se puede descubrir cómo funciona realmente el código, si funciona conforme a los requisitos y propósitos para los que fue diseñado y además permite encontrar algún comportamiento errático o inesperado.

Esta fase puede ser complicada si no se aplican metodologías que favorezcan las pruebas constantes sobre el código, para nuevos desarrollos o para proyectos ya empezados. La mayoría de las metodologías de desarrollo de software ágiles y tradicionales cuentan con fases para el testeo del código desarrollado, pero muchas veces estas fases no se llevan a cabo correctamente.

Teniendo en cuenta este problema, la compañía, viendo las necesidades de sus clientes y del mercado, propuso la idea de desarrollar una tecnología que pueda ser integrada en equipos de desarrollo para poder ejecutar pruebas unitarias, de integración y funcionales en la nube.

Con la misma idea en mente, se planteó también la posibilidad de generar o usar un servicio capaz de recuperar los resultados de dichas pruebas y generar un informe. Esto añadiría mucho valor al servicio desde el punto de vista de la administración del proyecto; ya que se podría tener una visión general de cómo va el desarrollo, donde están las flaquezas del código, comprobar si los distintos equipos de desarrollo están rindiendo por debajo de la esperado, etc.

2. Estado actual y objetivos iniciales

2.1. Estado Actual

Actualmente, la compañía se encuentra con que algunos de sus clientes tienen varios problemas en sus equipos de desarrollo. Una de las dificultades más graves que se encuentra son el desarrollo de pruebas, que suelen ser deficientes o muy escasas. Uno de los equipos que más sufría este problema es el de desarrollo de aplicaciones *iOS,* ya que no existen muchos servicios para agilizar las pruebas en dicha plataforma.

Cuando comenzó el proyecto, al autor de este documento se le asignó el desarrollo del servicio en *iOS* y *MacOS*, con la meta de generar un servicio que funcionara bajo el *IDE* de programación de *Apple*, *Xcode* y también bajo su lenguaje *Swift*. Tras una investigación sobre otros productos en el mercado, no se encontró ningún servicio que ofreciera la flexibilidad que se busca en ambos sistemas operativos (*iOS* y *MacOS*). La mayoría de las herramientas que se encuentran están centradas sobre todo en el desarrollo de aplicaciones para teléfonos inteligentes como, por ejemplo*: fastlane, testflight* o *bitrise*. Estas herramientas aportan una gran ventaja a la hora de desplegar las aplicaciones dentro de la tienda *Apple Store*, pero no proporcionan la funcionalidad que se busca con este proyecto.

2.2. Objetivos

Conociendo bien el problema, se plantea el desarrollo de un servicio que sea capaz de integrarse en varios proyectos de desarrollo bajo el entorno de *iOS* y *MacOS*. Además, deben ejecutar una serie de pruebas con la meta de probar la calidad, robustez y otras características del código.

Los objetivos serían los siguientes:

- La realización de una canalización que sea capaz de, en distintas fases, ejecutar pruebas bajo el entorno de desarrollo de *Apple*.
- Generar un servicio que sea capaz de integrarse fácilmente, y sin mucho esfuerzo, con varios equipos de desarrollo.
- Conseguir una integración continua con los distintos repositorios.
- Que sea capaz de ejecutar distintos tipos de pruebas (unitarias y funcionales) en un entorno aislado en la red.
- Que tenga la capacidad de recuperar al final de la ejecución de las pruebas los resultados de estas, que estos resultados sean visibles en un panel de control y generar un informe con esta información.
- Recoger información como etiquetas y descripciones de las pruebas ejecutadas para enriquecer el informe.

2.3. Estado del arte

Como se ha ido hablando anteriormente en este documento, las pruebas en algunos entornos de desarrollo suelen ser deficientes o más bien nulas. Ahí es donde entra este proyecto, que resuelve una necesidad dentro del mercado de la programación y la automatización que no se está explotando.

Al investigar en el mercado productos que cumplan las características que se buscan, no se encuentra las funcionalidades que se proponen con este proyecto. Sobre todo, existen servicios que buscan suplir las necesidades de integración y despliegue continuo. Esto en un principio permite ejecutar las pruebas del código, sin embargo, no da la posibilidad de generar un informe con los resultados de estas pruebas.

Un ejemplo de esto es *BitRise*, una plataforma dedicada a la integración y despliegue continuo de aplicaciones, donde se puede generar un flujo de trabajo para lograr ejecutar las pruebas que se le indique en el código. Estas son tareas muy útiles que necesitaremos para desarrollo de nuestro proyecto, pero no cuenta con ninguna forma de generar un informe con la información relevante de las pruebas.

Lo mismo ocurre con *CircleCI*, una plataforma parecida. Cuenta con varios servicios disponibles para poder generar un flujo de trabajo personalizado para distintos proyectos. Al igual que la anterior, no se encuentra la forma de generar informes sobre la ejecución de las pruebas.

También existen servicios como *Fastlane,* que se encuentra más orientado al despliegue de las aplicaciones dentro de la tienda de *Apple*. Finalmente, el servicio *TestFlight,* que se orienta a probar aplicaciones en otros dispositivos de *Apple* para comprobar el funcionamiento real.

3. Metodología, planificación y herramientas

3.1. Metodología

Como metodología se ha escogido una metodología ágil como es *SCRUM*, ya que aporta una gran capacidad de respuesta al cambio, flexibilidad y rapidez. Esta metodología es usada también en otros proyectos de la compañía ya conocidos, así que será mucho más fácil aplicarla en este desarrollo.

SCRUM es uno de los modelos de las metodologías ágiles más famosas. Sus bases nacen de los doce principios del manifiesto ágil, de entre los cuales destaca: la aceptación de los cambios de requisitos por parte del cliente, que el desarrollo gire en torno a trabajadores motivados, el software funcional como medida de avance en el proyecto y la simplicidad.

Siguiendo el marco de trabajo de la metodología *SCRUM,* existen 5 roles que desempeñan distintas funciones:

- *Product owner*: Es la persona dentro del equipo de desarrollo que representa al cliente, conoce bien sus necesidades y toma decisiones dentro del desarrollo.
- *Scrum master*: Es la persona que facilita, organiza y gestiona el trabajo dentro del equipo de desarrollo.
- Equipo: Es el grupo de personas que desarrolla el producto final mediante la entrega de incrementos funcionales.
- Usuario: El grupo de personas que usarán el producto que genere el equipo.
- *Stakeholder*: Son los clientes, su función es definir los requisitos además de dar retroalimentación sobre el producto.

Además, el marco de trabajo de *SCRUM* también define cómo debería ser el proceso iterativo que se va a seguir durante el desarrollo. Se basa en la entrega de un producto funcional, cada periodo de tiempo definido por el equipo (entre dos y cuatro semanas). Ese periodo se llama *sprint*, y es el tiempo en el que el equipo desarrolla el incremento.

Las funcionalidades que se deseen desarrollar se verán definidas por el *product owner* en el *product backlog* y se ordenan según prioridad. Las funcionalidades que el equipo deba desarrollar y entregar durante el *sprint* se encuentran en el sprint *backlog*, que son los objetivos que el equipo, *el scrum master* y el *product owner* han decidido añadir para ese *sprint*.

En la siguiente ilustración se puede ver de una forma más gráfica.

Ilustración 1: Diagrama funcionamiento metodología SCRUM

3.2. Planificación del proyecto

La planificación inicial que se realizó para este trabajo fin de título se muestra en la siguiente tabla:

Fases	Duración Estimada(horas)	Tareas
Estudio previo / Análisis	30	Tarea 1.1: Estudio de los servicios de *AWS*
		Tarea 1.2: Análisis de la compatibilidad entre *ReportPortal* y *Xcode*
Diseño / Desarrollo / Implementación	180	Tarea 2.1: Implementación de la canalización del repositorio
		Tarea 2.2: Implementación de la fase de compilación y pruebas
		Tarea 2.3: Despliegue de *ReportPortal*
		Tarea 2.4: Conexión de las pruebas con el *ReportPortal*
		Tarea 2.5: Creación del informe
Evaluación / Validación / Prueba	20	Tarea 3.1: Pruebas con varias aplicaciones
		Tarea 3.2: Validación de las funcionalidades
Documentación	70	Tarea 4.1: Documentación de los resultados y el proceso
		Tarea 4.2: Preparación de la defensa

Tabla 1: Planificación original TFT

En la planificación se puede ver como se tenía la intención de dedicar 30 horas al estudio y al análisis del proyecto, centrándonos en los servicios de *AWS, ReportPortal* y el propio *IDE* de *Apple, Xcode*. El diseño, desarrollo e implementación llevaría 180 horas, que se dedican a las cinco principales tareas. Se invertirían 20 horas en la validación de las aplicaciones que usen distintas formas de instalar el *plugin* del *ReportPortal* (como *framework* dentro del proyecto y como *pod* de *cocoapods*). Por último, se planteó dedicar 70 horas al proceso de elaboración de este documento y a la preparación de la defensa del proyecto.

Sin embargo, durante el desarrollo de este proyecto se encontró la necesidad de realizar ciertos ajustes a la planificación realizada, ya que se descubrieron ciertas dificultades inesperadas, como la compatibilidad de *ReportPortal* con *Xcode*. La integración del *ReportPortal*

con *Swift* y *Xcode* dio muchos problemas por estar desactualizada; la *API* por la cual funciona *ReportPortal* ha sido actualizada con el tiempo y no se ha mantenido igual de actualizada la integración. Debido a estos motivos surgió la necesidad de actualizar la implementación del *framework*, lo cual llevó más tiempo de lo esperado. Además, se aprovecharó para mejorar la implementación inicial y añadir más funcionalidades que podían ser interesantes para el proyecto.

Por estos motivos, se tuvo que añadir dos tareas nuevas a la planificación. La primera, de análisis, en la que investigaría cómo se tiene que integrar *ReportPortal* con el *framework* para ver qué es lo que falla y entender cómo funciona. La otra tarea sería la implementación, para aplicar los cambios necesarios al *framework* y que pueda funcionar la integración con *ReportPortal*, además de añadir nuevas funcionalidades.

Adicionalmente, se eliminó uno de los criterios de validación, a falta de más aplicaciones donde probar el proyecto y por el tiempo que ocuparon las nuevas tareas. Se reasignaron esas horas al desarrollo de las nuevas tareas. Finalmente, tras las diversas modificaciones, la ejecución del proyecto quedó desglosada tal y como se muestra en la siguiente tabla:

Fases	**Duración Estimada(horas)**	**Tareas**
Estudio previo / Análisis	30	Tarea 1.1: Estudio de los servicios de *AWS*
		Tarea 1.2: Análisis de la compatibilidad entre *ReportPortal* y *Xcode*
		Tarea 1.3: Análisis de la integración con el *ReportPortal*
Diseño / Desarrollo / Implementación	190	Tarea 2.1: Implementación de la canalización del repositorio
		Tarea 2.2: Implementación de la fase de compilación y pruebas
		Tarea 2.3: Despliegue de *ReportPortal*
		Tarea 2.4: Mejorar implementación *framework* con *ReportPortal*
		Tarea 2.5: Conexión de las pruebas con el *ReportPortal*
		Tarea 2.6: Creación del informe

Evaluación / Validación / Prueba	10	Tarea 3.1: Validación de las funcionalidades
Documentación	70	Tarea 4.1: Documentación de los resultados y el proceso
		Tarea 4.2: Preparación de la defensa

Tabla 2: Planificación final TFT

3.3. Herramientas

Amazon Web Services

Ilustración 2: Amazon Web Services

Amazon Web Services (Ilustración 2) es la plataforma de servicios de computación en la nube del gigante tecnológico *Amazon*. Ofrece una gran variedad de servicios (más de 200) entre los que se encuentran el análisis y almacenamiento de datos, bases de datos, *machine learning*, aplicaciones y funciones sin servidor (*serverless*), por poner algunos ejemplos. Con todos estos servicios, *Amazon Web Services* se sitúa entre uno de los mejores proveedores de servicios de computación en la nube, cubriendo casi todas de las necesidades de la mayoría de los clientes. Si esto no fuera poco, también destaca por la gran comunidad de clientes y socios que tiene, entre los que se podrían destacar *Netflix*, *AstraZeneca* y *ARM*. También sobresale por lo rápido que incorporan nuevas tecnologías y añaden nuevos servicios, además de por la cantidad de centros de computación que tiene situados alrededor del mundo para tener siempre el mejor rendimiento sin depender de donde se esté geográficamente.

Garnet, que es una compañía líder dentro de la consultoría e investigación en el campo de las tecnologías de la información, publicó, el pasado julio del año 2021, un informe en el que se comparaban los distintos servicios de infraestructura y de plataforma en la nube. En dicho informe se puede leer y ver, gracias al cuadrante mágico de la ilustración 3, cómo *Amazon Web Services* es líder del mercado en gran parte gracias a sus servicios y sus innovaciones.

Ilustración 3: Cuadrado Magico de Garnet

Por todos estos motivos se escogió *Amazon Web Services*. Aun así, dentro de estos servicios, para el desarrollo de la canalización, decidimos usar los siguientes.

CodePipeline

CodePipeline (Ilustración 4) es otro de los servicios de *Amazon Web Services*. Está orientado completamente a la integración y distribución continua (*CI/CD*) de diferentes proyectos. Sirve sobre todo para automatizar tareas de distribución de aplicaciones como pueden ser fases de pruebas, de compilación y también de implementación.

Ilustración 4:AWS CodePipeline

Destaca sobre todo por lo fácil e intuitivo que hace el proceso de integración y distribución continua. Consta de una serie de fases que no son limitadas y que se pueden ejecutar de forma totalmente libre. Estas fases son altamente personalizables para el proyecto en cuestión. Por ejemplo, se pueden hacer distintos tipos de compilación para diversos entornos con configuraciones totalmente diferentes para el mismo proyecto. También se podría generar varias fases de despliegue, bajo diversos entornos, para subir una misma aplicación a las tiendas de *Apple* y de *Android*.

Dentro de esta canalización se ejecutarán las dos siguientes herramientas: *CodeStarSourceConnection y CodeBuild.*

CodeStarSourceConnection

CodeStarSourceConnection es otro de los servicios de *Amazon Web Services* que se usarán en este proyecto. Este servicio permite conectar la canalización con la herramienta de control de código fuente de tipo *Git*. En este caso se usará *BitBucket* como herramienta de control de código fuente.

CodeStarSourceConnection también se usa dentro de la canalización como si fuera una fase más de la misma. El papel de este servicio en el proyecto será el de acoplarse al repositorio que se elija, de forma que cada vez que haya una actualización dentro del repositorio se empiece a ejecutar la canalización con el nuevo código actualizado.

CodeBuild

CodeBuild (Ilustración 5) es el servicio de *Amazon Web Services* que se usará para poder ejecutar las pruebas. Como el anterior servicio, puede ser una fase más del *CodePipeline*. Eso da la capacidad de añadir tantas fases de compilación como sean necesarias, además de dar la posibilidad de personalizar cada una.

Destaca también por la gran capacidad de escalar los recursos de una manera bastante sencilla, el aislamiento de las maquinas que compilan entre sí y la capacidad de monitoreo constante de las distintas instancias. *CodeBuild* tiene una gran variedad de sistemas operativos y entornos para poder ejecutar las fases de compilación.

Ilustración 5: AWS CodeBuild

S3

El servicio *Amazon Simple Storage Service* (Ilustración 6) es el sistema de almacenamiento en la nube de *Amazon*. Durante este proyecto no se configurará, pero se usará para alojar los resultados de las distintas fases de la canalización. Este servicio es muy demandado por una

gran cantidad de organizaciones debido a su gran escalabilidad, sus funciones de copias de seguridad, su altísima fiabilidad y su disponibilidad.

Ilustración 6: AWS S3

EC2

Amazon Elastic Compute Cloud (*Amazon EC2*) es una plataforma que ofrece computación en la nube. Es perfecto para poder ejecutar las pruebas en un ecosistema tan hermético como es el de *Apple*. *EC2* (Ilustración 7) se caracteriza por lo altamente personalizable que es. Cuenta con más de 500 instancias distintas dando también la posibilidad de elegir procesadores, almacenamiento, sistemas operativos y memoria. Como la mayoría de servicios de la compañía tiene una gran capacidad de escalado bajo demanda y seguridad.

Ilustración 7: AWS EC2

Para poder ejecutar y compilar aplicaciones para *iOS* se necesita un dispositivo de *Apple* como por ejemplo, un Mac. *EC2* es el único servicio de computación de la nube que tiene instancias de *Mac* (*Mac Metal*) bajo demanda.

AWS IAM

El último servicio de *AWS* que se usará es el *AWS Identity and Access Management* (Ilustración 8). Este se encarga de darle permisos a los usuarios y procesos, para poder acceder a los distintos servicios y recursos. Se basa en un sistema de políticas, las cuales se van asignando a los distintos procesos, máquinas, instancias y usuarios para darles accesos específicos a

determinados recursos y servicios. Gracias a esto se genera un gran control sobre quién usa qué recursos.

Ilustración 8: AWS IAM

Jenkins

Jenkins (Ilustración 9) es una aplicación de código abierto, basada en *Java*, de automatización de tareas de integración y despliegue continuos. Normalmente se ejecuta en un servidor con la idea de automatizar la mayoría de las tareas de compilación, integración y despliegue necesarias. Es personalizable para que sea capaz de ejecutar lo que se le programe en cualquier tipo de orden. Cuenta con una serie de *plugins* que se pueden utilizar para expandir sus funcionalidades.

Ilustración 9: Jenkins

Se puede intercalar como una fase más del servicio *CodeBuild* de *AWS* y tiene soporte en varias plataformas como puede ser *Linux*, *MacOS*, *Windows*, *Docker*, *Kubernetes* y otros. En este caso se usará en la instancia *EC2* de *MacOS*.

ReportPortal

ReportPortal (Ilustración 10) es otra aplicación de código abierto, que se usa como panel de control automatizado con los resultados de las pruebas de los proyectos. Como ventajas permite analizar las pruebas en tiempo real, visualizar los resultados y métricas de manera muy sencilla e intuitiva. Cuenta con una fácil integración con varios *framework*s orientados a

las pruebas, lleva un registro de cómo evoluciona el proyecto y además cuenta con la capacidad de añadir registros individuales a algunas pruebas para enriquecer la información. Se puede desplegar en *Docker*, *Kubernetes* y en *Linux*.

Ilustración 10: ReportPortal

Xcode

Xcode (Ilustración 11) es el entorno de programación desarrollado por *Apple* para aplicaciones *iOS, MacOS, tvOs* y *watchOS*. Soporta varios lenguajes (*Objective-C, Swift, C#, Python, AppleScript, C++, HTML5 y Java*). Es esencial a la hora de desarrollar aplicaciones para el ecosistema de *Apple* y todas sus funcionalidades solo son accesibles desde de un equipo con sistema operativo *MacOS*.

Ilustración 11: Xcode

Este *IDE* da al desarrollador un conjunto de herramientas muy completas, como pueden ser funcionalidades para el diseño de interfaces de usuario, emulación de dispositivos, pruebas, depuración, subida de aplicaciones a la *AppStore*, *testflight*, herramientas para la administración del código fuente y compilación de aplicaciones.

BitBucket

Es un servicio (Ilustración 12) de control de código fuente alojado en la web que usa *Git*. Una de las ventajas que tiene es su integración a la perfección con el resto de los servicios de *Atlassian*. En concreto, con *Jira*, para tener un registro de las ramas al poder asignarlas a

tareas, para poder tener un mejor control del desarrollo del trabajo. También permite hacer revisiones de código y *pull request*. Es una herramienta ampliamente usada por distintas compañías, gracias a sus integraciones con servicios de terceros, como puede ser *AWS*.

Ilustración 12: Bitbucket

Docker

Docker (Ilustración 13) es una herramienta de virtualización por contenedores, de código abierto, usada mayoritariamente para el despliegue de servicios y aplicaciones web. Las ventajas de este tipo de virtualización, comparado a la tradicional, serían: la portabilidad que permite a la hora de mover contenedores sin tener problemas de compatibilidad, lo poco demandantes que pueden ser los contenedores al no tener que emular todo el hardware y la escalabilidad que esto ofrece.

Ilustración 13: Docker

4. Justificación de las competencias específicas cubiertas

4.1. TI01

"Capacidad para comprender el entorno de una organización y sus necesidades en el ámbito de las tecnologías de la información y las comunicaciones."

Este proyecto siempre se planteó comprendiendo la estructura e infraestructura de la compañía, además de las necesidades de la empresa para poder mejorar el servicio o incluso optimizarlo a su ámbito de las tecnologías de la información y las comunicaciones.

4.2. TI02

"Capacidad para seleccionar, diseñar, desplegar, integrar, evaluar, construir, gestionar, explotar y mantener las tecnologías de hardware, software y redes, dentro de los parámetros de coste y calidad adecuados."

Durante el desarrollo de este proyecto se escogieron las herramientas y los servicios que eran necesarios, teniendo en cuenta los recursos y las capacidades de la compañía para buscar una solución satisfactoria para ser capaces de resolver el problema que se había planteado.

4.3. TI05

"Capacidad para seleccionar, desplegar, integrar y gestionar sistemas de información que satisfagan las necesidades de la organización, con los criterios de coste y calidad identificados"

Esta competencia se desarrolló en su mayor parte en la fase de diseño. Se tuvo en cuenta la escalabilidad y los costes del sistema, por eso se escogieron servicios que funcionen bajo demanda para poder reducir los costes, sin llegar a comprometer el rendimiento final del servicio.

5. Aportaciones

La principal motivación de este proyecto fue, desde su comienzo, mejorar el desarrollo de software en equipos de desarrollo que, por desconocimiento, malas prácticas u otros motivos, no comenzaron aplicando pruebas a su código o no las aplican correctamente. Por ello se intentará mejorar en gran medida la calidad del código, la estabilidad y la robustez de este produciendo así mejores programas, servicios y productos.

También tiene valor desde el punto de vista de la administración del proyecto. Perfiles más analíticos pueden ver cómo va avanzando el desarrollo de las pruebas, junto a sus resultados y cómo el equipo está rindiendo. Ademas brinda la posibilidad de aplicar metodologías como el desarrollo guiado por pruebas, para favorecer la calidad del producto y mejorar el seguimiento del desarrollo.

Como se destacó al principio de este documento, existen varios servicios en el mercado, pero ninguno satisface el problema que se busca resolver. La mayoría están orientados a la automatización del proceso de despliegue de la aplicación dentro de la *App Store*. Teniendo esto en cuenta, este servicio cumple y resuelve unas necesidades que anteriormente no se resolvían en el mercado, dándole un gran atractivo y valor a este producto.

Pensando en el diseño del servicio, la elasticidad, personalización y la escalabilidad también aporta un valor añadido, ya que al producto se le puede añadir fácilmente nuevas funciones o enriquecer las que ya existen, lo cual se desarrollará en el apartado ocho de este documento.

6. Legislación y Licencias

Legislación

Dentro de la legislación no hay nada que destacar salvo la protección y tratamiento de los datos. Los únicos momentos en el que se usarán datos personales será a la hora de enviar un correo electrónico desde *Jenkins* y al enviar una invitación a un correo electrónico en *ReportPortal,* para poder generar una cuenta dentro del servicio.

Todos estos datos personales se tratarán de acuerdo con la Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y Garantía de los Derechos Digitales (LOPDGDD). Estos datos solamente serán usados por *Jenkins* para mandar correos electrónicos en forma de notificaciones y tener un correo desde el que poder iniciar sesión en *ReportPortal*. Si se piensa en ámbito europeo, el Reglamento General de Protección de Datos (RGPD) asigna a estos tipos de datos el nivel básico.

Licencias de software libre

Licencia MIT

La licencia MIT es una licencia creada por el Instituto Tecnológico de Massachussets que ha sido ampliamente usada en el campo del desarrollo de software libre, pero también puede ser usada para software privado. Da las posibilidades de modificación, uso comercial, uso privado y su distribución.

El agente de *ReportPortal* y *Jenkins* hacen uso de esta licencia.

Apache 2.0

La licencia Apache 2.0 es una licencia de software abierto creada por la fundación *Apache Software*. La principal diferencia con la anterior es que no es *copyleft,* así que no exige que las copias derivadas usen la misma licencia. En cuanto al resto de características es bastante parecida, muy permisiva, permitiendo la distribución y modificación para cualquier uso.

Las herramientas que usan esta licencia en este proyecto son *ReportPortal* y *Docker*.

Licencia GPL

La licencia GPL es la otra licencia de código abierto creada por Richard Stallman y por la *Free Software Foundation*. Permite el uso comercial, la modificación, la distribución, el uso de patente y el uso privado. También destaca por la obligación de que las posibles modificaciones tengan que usar la misma licencia.

Esta licencia la usa *Git* en el proyecto.

Licencias de software privado

Hay que destacar que todas las licencias de software de pago fueron cedidas por *Advance Digital Experts*.

AWS

Amazon Web Services no tiene ninguna licencia como tal, pero todos sus servicios se pagan bajo demanda y por su uso.

Apple Developer Account

Para usar todas las herramientas que brinda *Apple* a la hora de desarrollar se necesita una cuenta en *Apple Developer*. Esta cuenta se puede conseguir de forma gratuita, para tener acceso a muchos servicios y las herramientas, pero a la hora de compilar o publicar en la tienda sí que haría falta pagar anualmente una cuota.

Esta licencia es usada por *Xcode*.

7. Desarrollo

Durante este apartado se explicarán todas las tareas que se tuvieron que completar para la realización de este proyecto. Como se explicó en el apartado 2.2, el objetivo final es la generación de una canalización capaz de adherirse a un repositorio, escuchar una rama y que cada vez que haya un evento en esa rama, se ejecuten ciertas tareas, como la ejecución de las pruebas.

7.1. Estudio Previo

Lo primero que se hará es una fase de estudio previo con la idea de informarnos, aprender y familiarizarnos con los servicios y herramientas que se van a usar. No se tiene experiencia con ninguna de las herramientas de las que se ha hablado durante este documento a excepción de *Docker*, así que por eso se cuenta con una fase de estudio previo. Se analizarán los servicios de *Amazon Web Services*, luego *Xcode*, *ReportPortal* y por último la integración de *ReportPortal* y *Xcode*.

7.1.1. Estudio de los servicios de *Amazon Web Services*

El primer servicio de la suite de productos de *Amazon Web Services* que se estudiará es *CodePipeline*, seguido por *CodeStarSourceConnection*, *EC2*, *S3* e *AWS IAM*.

CodePipeline

CodePipeline es el servicio que permite generar canalizaciones para ayudar con las necesidades de integración y entrega continua de este proyecto, además de dar la oportunidad de automatizarlas.

Con la ayuda de este servicio se generarán canalizaciones, las cuales no son más que flujos de trabajo predefinidos basados en etapas y acciones, como se puede ver en la ilustración 14. Las etapas son, las fases o los pasos que siguen las canalizaciones. Estas ayudan a aislar los entornos entre distintas etapas. Las etapas reciben, crean, usan, y modifican artefactos que no son más que elementos dentro de la canalización que pueden ser de varios tipos. Un ejemplo de artefacto puede ser un *commit* a una rama de un repositorio. En el caso de una etapa, podría ser cuando se compila o se ejecuten pruebas sobre esa rama. También otro ejemplo puede ser una etapa que coja ese artefacto y lo despliegue en un entorno preconfigurado. Las distintas etapas se interconectan por transiciones.

Todas las etapas las forman un conjunto de una o más acciones, las cuales pueden ser ejecutadas tanto en serie como en paralelo. Las acciones no son nada más que procesos u operaciones que se realizan en un orden especifico. Estas acciones pueden ser la iniciación de la canalización por un cambio en la rama del repositorio o la ejecución de un comando de compilación bajo un entorno en concreto, por ejemplo.

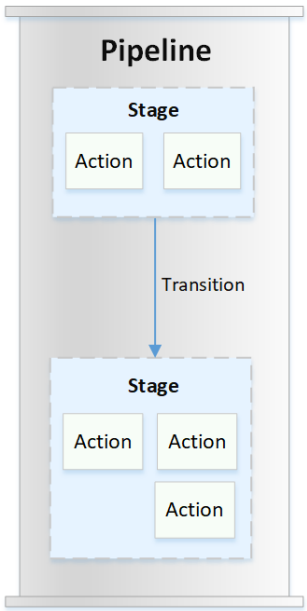


Ilustración 14: Diagrama de una canalización

Estas acciones pueden ser de distintos tipos: fuente, compilación, pruebas, despliegue aprobación e innovación. Dentro de cada uno de los tipos se encuentran distintos

proveedores. Las acciones de tipo fuente cuentan como proveedores: *Amazon S3*, *Amazon ECR*, *CodeCommit* y *CodeStartSourceConncection*. Los de tipo compilación serían: *CodeBuild, Custom CloudBeed, Custom Jenkins y Custom Team City*. Existen nueve proveedores de pruebas, entre los cuales hay que destacar los siguientes: *AWS Device Farm, Custom BlazeMeter, CodeBuild* y *Custom Jenkins*. Como el anterior caso hay hasta diez proveedores de implementación, entre los que destacan: *Amazon ECS, AmazonAlexa, CodeDeploy, AWS CloudFormation* y *Amazon OpsWork*. Como proveedor de aprobación solo existe *manual*. Por último, los proveedores de innovación solo existen dos: *AWS Lambda* y *AWS Step Functions*.

Cada ejecución de una canalización es única y con ella se genera un identificador por ejecución. Dichas ejecuciones pasan de forma ordenada por las distintas etapas y fases de la canalización generando una serie de cambios y artefactos por etapa. Solo puede existir una ejecución de canalización por etapa a la vez, pero pueden haber más de una por canalización. Si una ejecución de canalización está esperando a entrar a una etapa que está ocupada, se le llama "ejecución entrante" y estas son especiales, ya que a diferencia del resto no pueden fallar, detenerse o sustituirse de forma manual. Los posibles estados de las ejecuciones de canalización son: *InProgress, Stopping, Stopped, Succeeded, Superseded* y *Failed*.

También existen las ejecuciones de acción que comparten muchas características con las ejecuciones anteriores, como la asociación de un identificador único a cada una. La diferencia es que estas ejecutan operaciones sobre los artefactos asignados a las etapas en la que se encuentran. Los estados posibles que tienen son: *InProgress, Abandoned, Succeeded* y *Failed*.

CodeStarSourceConnection

Esta herramienta se usará para establecer una conexión entre la canalización y el repositorio del proyecto final. También se usará para iniciar la ejecución de la canalización. Funciona de tal forma que instala un *webhook* en nuestro servicio de *Git*. Con este *webhook* escucha cada vez que tiene lugar un evento de *Git* en una rama que se le indique. Tras este evento, lo que hace es recoger todo el código fuente de esta rama y lo traslada a la canalización como un artefacto y empieza la ejecución de la canalización.

Los servicios de terceros que se pueden usar con esta herramienta son: *BitBucket Cloud, Github Cloud* y *Github Enterprise Server*. También existen dos tipos de copias a la hora de coger el código del repositorio, una que solo copia el último *commit*, y otra que es llamada clonación completa, que también recoge los metadatos y carpetas ocultas.

CodeBuild

CodeBuild es la herramienta que permite implementar tareas de integración continua dentro de la canalización. Como la anterior herramienta, esta puede ser usada como una acción más en la canalización. Escala de forma automática y es capaz de administrar varios procesos a la vez. La herramienta se configura de tal forma que dentro de cada etapa se le asigna los artefactos a usar, el nombre de los artefactos de salida y un proyecto de compilación.

El proyecto de compilación es lo que se utiliza para indicar a *CodeBuild* lo que tiene que ejecutar. Estos proyectos son altamente configurables y se dedican a procesar el artefacto como se le indique para devolver un resultado en forma de artefacto salida. Un ejemplo puede ser la compilación de un proyecto de *iOS* y que acabe devolviendo, como artefacto de salida, el archivo *IPA* compilado.

Los proyectos de compilación se pueden configurar para ejecutarse bajo imágenes administradas por *AWS CodeBuild,* entre las que se encuentran *Ubuntu* y *Amazon Linux* 2 o también se puede ejecutar bajo imágenes personalizadas. Con esta opción se puede ejecutar *Docker* en varios entornos como puede ser *ARM* o *Linux*. Bajo estas imágenes se pueden ejecutar y desplegar infinidad de servicios, se puede lanzar una *API REST* con *Docker*, se puede alojar un servicio web o usarlo con el mismo objetivo para la integración continua, pero para otros lenguajes de programación.

Es importante recalcar que se puede configurar la capacidad de memoria *RAM* y de núcleos lógicos para mejorar el rendimiento del servicio o para bajar estos recursos en caso de que no sean aprovechados. Además, es posible configurar un tiempo de ejecución y de espera máximo.

La forma en la que se definen los pasos o los procesos que tienen que ejecutar estos proyectos de compilación, se hace usando un archivo *BuildSpec*. Esto es un archivo de configuración que se alojaría en el repositorio. Lo conforman varios campos de los cuales se destaca: *version, run-as, env* y *phases (install, pre_build, build* y *post-build)*. Dentro de estas fases se pondrían explícitamente los comandos a ejecutar dentro de un campo llamado *commands* y *artifacts*.

EC2

Amazon Elastic Compute Cloud o *Amazon EC2* es un servicio de computación en la nube, altamente escalable y configurable. Permite así evitar la compra de hardware físico evadiendo los gastos que conllevan la compra de este: gastos por su mantenimiento, gastos energéticos y gastos de administración. Lo conforman instancias que no dejan de ser entornos

informáticos virtuales alojados en la nube. Estas instancias no es necesario configurarlas desde cero, este servicio consta de instancias preconfiguradas llamadas *Amazon Machine Image* (*AMI*) que ya constan de sistema operativo, además de software adicional. Como se ha indicado anteriormente, *Amazon* tiene varias ubicaciones físicas alrededor del mundo, donde poder alojar las instancias. De esta manera se puede conseguir una mejor latencia cuando se usan, al escoger una localización que esté cercana a donde trabajamos o donde se usa el servicio.

También permite la configuración de los recursos para cada instancia, usando lo que llaman tipos de instancia. Son las configuraciones disponibles donde varían los procesadores, la cantidad de núcleos y la arquitectura de estos (*x86* y *ARM*), la cantidad de memoria, el almacenamiento y las capacidades de red. Consta de volúmenes de almacenamiento persistentes y volátiles que se eliminan cuando la instancia se termina, se apaga o se hiberna. Por último, añadir que existen tipos de instancias con *GPU,* las cuales son ideales para tareas de *machine learning,* de inteligencia artificial y para el renderizado de contenido audiovisual.

Las instancias también destacan por la seguridad. Las credenciales para el inicio de sesión son pares de claves, las cuales *AWS* almacena la llave pública. La primera vez que se configura cede acceso a la clave privada para que se pueda guardar en un lugar seguro. Además, permite configurar un cortafuegos para especificar qué protocolos, puertos y direcciones *IP* tienen autorización para establecer cualquier tipo de conexión a la instancia. Así mismo, se pueden definir *Virtual Private Cloud* (*VPC*), que son redes virtuales privadas que dan la oportunidad de agrupar instancias bajo una misma red. Estas redes pueden estar aisladas del resto o no.

AWS también consta de servicios para poder completar las instancias *EC2*, como el *EC2 Auto Scaling*, que sirve para poder aumentar el número de las instancias si la aplicación tiene mucha carga de trabajo. Aquí se le pueden asignar un numero de instancias mínimas y un máximo para que vaya escalando hacia arriba o hacia abajo dependiendo de la carga. Al mismo tiempo también ofrece *AWS CloudWatch* que permite monitorizar la mayoría de las estadísticas de las instancias en tiempo real. Estas estadísticas pueden ser: uso de procesador, memoria, almacenamiento, estado de esta o consola. Todas estas estadísticas pueden ser observadas bajo un mismo panel de control.

S3

En los anteriores servicios se han mencionado los artefactos, son los elementos de entrada y salida de los procesos. Estos artefactos tienen que ser alojados o guardados de alguna forma:

aquí es donde entra *S3*. *AWS S3* es un servicio de almacenamiento en la nube que destaca por lo sencillo que es de usar.

Dentro de *S3* existen varios tipos o clases de almacenamiento:

- *S3 Standard* es recomendado para datos a los que se necesita acceder cada poco tiempo o con bastante frecuencia.
- *S3 Standar-IA* sirve para datos a los que se acceden con menor frecuencia y así se ahorra en costes.
- *S3 Glacier* está más destinado a almacenamiento de datos de larga duración y con baja frecuencia de acceso.
- *S3 Intelligent-Tiering*, es un tipo especial de *S3*. Consta de 4 niveles de forma que el tiempo que se tarda en acceder a los archivos es variable según el nivel en el que se encuentre. El almacenamiento va aprendiendo los patrones de qué datos se abren y cada cuánto tiempo, así va distribuyendo los datos entre los niveles según considere con el objetivo de ahorrar costes.

Otra característica de *S3* es lo fácil que es administrar datos, siendo capaz de hacerlos públicos para poder acceder a ellos por la red. Además, permite poder bloquear el acceso a ciertos a ciertos servicios dentro de *AWS* o bien establecer un ciclo de vida para los datos, por poner algunos ejemplos. Así mismo, permite ejecutar análisis con el objetivo de conocer mejor el uso de los datos y la utilización de ellos para poder optimizar el almacenamiento al máximo. Por último, se destaca lo fiable y seguro que es.

El servicio funciona mediante los *buckets,* que son los contenedores de información. La cantidad de objetos que se pueden guardar dentro de los *buckets* es ilimitada y se puede llegar a tener cien *buckets* por cuenta. Lo único que se necesita para poder crear un *bucket* es asignarle un nombre y la región donde se necesite crearlo. Los objetos son los archivos que se guardan en los *buckets*. Cuando se añade o se crea un objeto, a cada uno se le asigna un identificador o clave, que es único dentro de ese *bucket* y contiene una serie de metadatos. Con la combinación del *bucket*, el identificador del objeto y el identificador de versión se consigue identificar de forma única cada objeto.

El control de versiones permite conservar el mismo objeto pero en varias variantes. Se puede usar para recuperarlo o restaurarlo a un estado anterior, así se pueden evitar errores humanos. El identificador de la versión, como dice el nombre, es la forma en la que *S3* controla las distintas versiones de los distintos objetos.

Hay que destacar también la accesibilidad de *S3*. Es accesible mediante el *Software Development Kit* de *AWS*, mediante una *API REST*, *AWS Management Console* y mediante *AWS Command Line Interface*. Con la *API REST* se tiene la posibilidad de usar esta herramienta en multitud de proyectos y lenguajes de programación.

AWS IAM

AWS Identity and Access Management es la herramienta que permite administrar los accesos a los distintos servicios de *Amazon Web Services*. Brinda el poder de controlar, en tiempo real, quién está autenticado y qué servicios es capaz de utilizar.

Permite asignar accesos de forma muy detallada a cada usuario. Como por ejemplo, permitir el acceso a partes muy concretas de un servicio en concreto, para evitar problemas con la seguridad. Un posible caso de uso sería dar permisos solo de lectura a un determinado *bucket* en *S3*.

Por defecto, los usuarios y servicios se crean con los permisos mínimos, de forma que se les va añadiendo permisos según los van necesitando. Estos permisos se llaman políticas y no solo se les permite asignar a usuarios sino también a servicios en concreto, como por ejemplo a una instancia bajo *EC2* para que no sea capaz de acceder a *S3* por motivos de seguridad, o que una etapa de la canalización no sea capaz de usar *EC2* o de administrar la canalización.

Existen varios tipos de políticas además de las dos que se han comentado hasta ahora (políticas basadas en identidades y políticas basadas en recursos). También existen: políticas de sesión, lista de control de acceso, SCP para organizaciones y límites de permisos.

7.1.2. Análisis de compatibilidad entre *Xcode* y *ReportPortal*

Antes de estudiar la compatibilidad entre estos programas se va a explicar un poco más en profundidad que es cada uno y cómo funcionan ambos.

Xcode

Tal y como se ha indicado anteriormente, *Xcode* es el entorno de desarrollo de aplicaciones para el ecosistema de *Apple*, diseñado por la propia compañía. Lo conforman un conjunto de herramientas para facilitar el trabajo del desarrollador, incluyendo utilidades para todas las fases del desarrollo. Las herramientas que incluye son:

- *Xcode Cloud.*
- Un gestor de código fuente.
- Posibilidad de ejecución de pruebas.

- Editor de código, optimizado para *Swift*.
- Herramientas para la depuración.
- Emulador de dispositivos *iOS*.

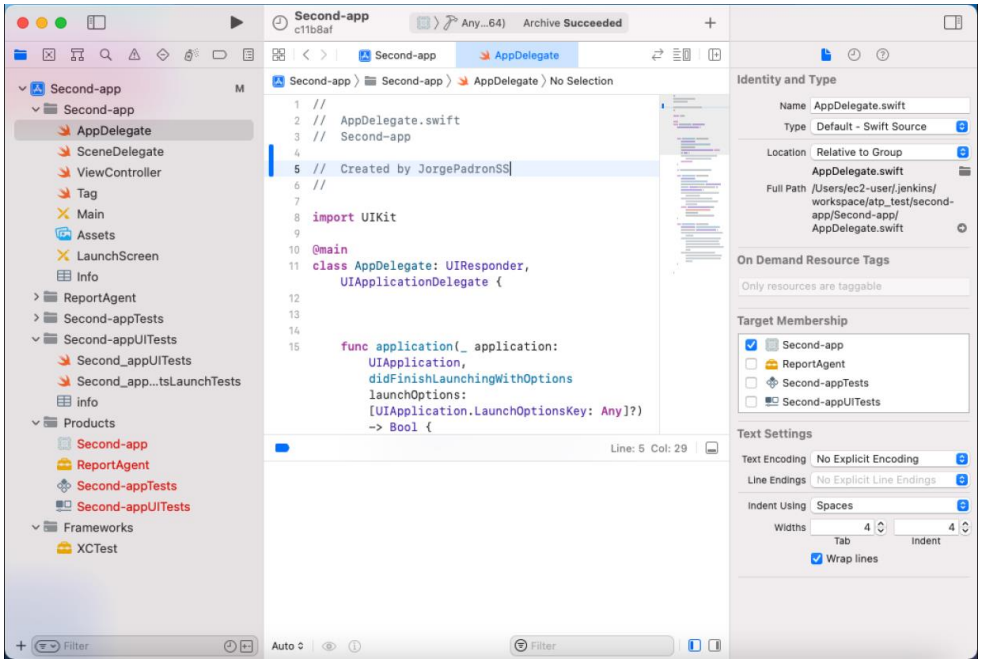


Ilustración 15: Xcode

En la ilustración 15 se puede ver el entorno de desarrollo. Lo conforman cinco áreas principales. En la parte superior se encuentra una barra de tareas que permite lanzar la ejecución de la aplicación, ver en qué estado se encuentra, qué se está haciendo ahora mismo y en qué dispositivo se quiere que se ejecute. En el lado de la izquierda se encuentra el área del navegador, donde se tiene una vista jerárquica de todos los archivos del proyecto. Los iconos de la parte de arriba de esta área representan distintas pestañas, como un buscador, una vista que recopila los avisos y los errores, una pestaña donde están todas las pruebas y una vista con los puntos de ruptura del código para el depurador. En la parte inferior se encuentra un área que está completamente destinada a la consola y una zona que sirve para la depuración. A la derecha se encuentra un área que brinda opciones de configuración. Si se encuentra en el archivo de configuración del proyecto, mostrará ajustes adicionales sobre este archivo. Si se edita una vista mostrará distintos ajustes del elemento que se esté editando, por ejemplo. Por último, en el centro se encuentra el área de edición, donde se puede seleccionar un archivo y escribir código, editar ajustes, añadir campos, etc.

Como se ha comentado, *Swift* es un lenguaje programación fuertemente tipado que se va a usar para este proyecto. Fue anunciado en 2014 en la conferencia de desarrolladores de *Apple*

y se creó para sustituir a *Objetive-C* dentro del mundo de desarrollo de aplicaciones *iOS*. Así mismo fue diseñado para integrase con *Cocoa* y *Cocoa Touch*. Dos años después de su lanzamiento, *Apple* decidió que se convirtiera en código abierto bajo licencia Apache 2.0.

Destaca por lo seguro que es, ya que al ser un lenguaje reciente fue diseñado con eso en mente y es capaz de detectar fácilmente los errores. Asegura ser rápido y eficiente, lo cual es importante ya que viene a sustituir lenguajes basados en *Objetive-C*. Por último, tiene una sintaxis muy expresiva que ayuda a entender mejor y más rápido el código, tendiendo a ser muy intuitivo.

Para crear y diseñar la interfaz gráfica de la aplicación se usará el *framework* diseñado por *Apple, SwiftUI*. Permite generar una aplicación más fácilmente. Cuenta con mejoras para *MacOS*, una gran variedad de opciones de accesibilidad y con él se será capaz de generar una aplicación con una interfaz moderna, rápida y robusta. Funciona mediante el uso de un modelo vista controlador, en el cual permite editar la vista prácticamente sin tener que escribir ni una línea de código. El controlador se usará para definir el nombre de los elementos de la vista, además de para añadir una capa de funcionalidad. Uno de los motivos por lo que se escogió es también por la gran integración que tiene con el resto de las herramientas de *Apple* y en concreto con las herramientas de pruebas.

Por el ámbito en el que se enmarca este proyecto se analizarán las pruebas dentro de *Xcode* y *Swift*. Dentro del desarrollo aplicaciones bajo el ecosistema de *Apple* no existen muchos *framework* orientados a las pruebas. El más popular es el propio de la compañía, *XCTest* que es el que se usará para este proyecto. Como única alternativa existe *Quick*, un *framework* que fue escrito en *Objetive-C* pensado para los desarrollos dirigidos por pruebas.

XCTest permite crear pruebas unitarias, de integración y de la interfaz gráfica sobre nuestro código. Se integra muy bien con *Xcode*, pudiendo tener una lista de pruebas donde se puede ejecutar una prueba sola sin tener que ejecutar todo el conjunto. Para poder lanzar las pruebas es tan sencillo como hacer clic en la prueba desde la pestaña pruebas del área del navegador o mantener pulsado el botón de ejecución y seleccionar lanzar con pruebas.

XCTest define lo que llama como *TestCase,* que no es más que una clase donde se alojan funciones con las pruebas o *TestMethods*. A los *TestCase* se le añaden un *Target*, que es la aplicación, o el código donde se va a probar las distintas funciones. El nombre de este debería describir o definir la funcionalidad que se va a probar. Dentro de las pruebas debería haber varios *assert*, que son funciones que ayudan a probar el código comprobando, por ejemplo,

si el contenido de dos variables es el mismo (*AssertEqual*) o simplemente probando que un valor booleano es verdadero (*assert*).

ReportPortal

ReportPortal es un servicio que se desplegará. Este servicio (Ilustración 16) da la capacidad de recabar toda la información de las pruebas una vez acabadas, mostrándolas en un panel de control. Esto brinda una capacidad de analizar todo lo que tenga que ver con las pruebas, rendimiento, tiempo de ejecución, si se pasan de forma satisfactoria o no, por poner algunos ejemplos. Dentro de la administración del proyecto y del negocio en si, toda esta información puede llegar a tener un gran valor.

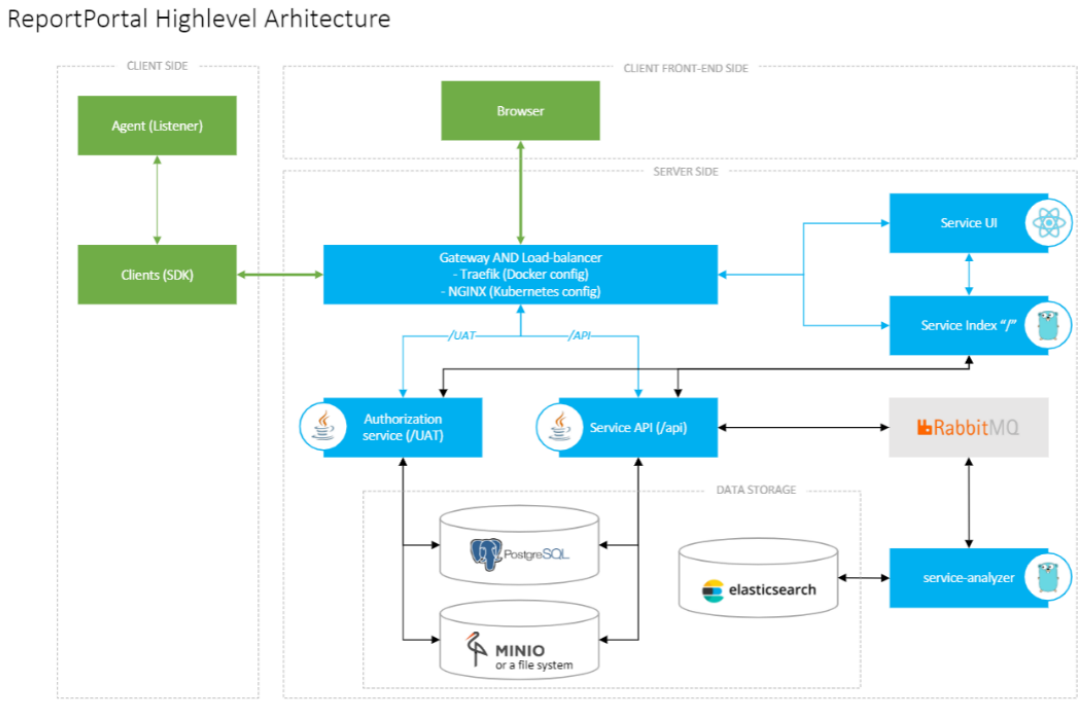


Ilustración 16: Arquitectura ReportPortal

El servicio funciona en tiempo real, así que permite ver cómo se van ejecutando las pruebas desde la propia plataforma, siendo capaz de ejecutar análisis automáticos para intentar buscar el origen del fallo de la prueba. Consta de integraciones con las principales plataformas y *frameworks* orientados a las pruebas.

La forma en la que se conecta a los distintos *frameworks* es a través del uso de distintos agentes, que son el software necesario para integrarse con *ReportPortal*. Estos agentes actúan de *Listeners* y van enviando a *ReportPortal* las pruebas, con sus resultados además de metadatos.

ReportPortal tiene agentes para *Java* (*Junit*, *Cucumber*, *Karate*, *Android Studio* y más), *.NET* (*Net Unit*, *VS Test*, *Specflow*, *Unit.net* y *Gauge*), *JavaScript* (*Mocha*, *Jasmine*, *Jest*, *TestCafé*, etc.), *python* (*pytest*, *RobotFramework*, *Python Behave* y *Nose*), PHP (*behat*, *phpUnit* y *Codeception*) y otros como pueden ser *XCTest*, *Silktest* y *Gwen*.

Cabe destacar que algunos de estos agentes los han desarrollado y mantenido el propio equipo de *ReportPortal*. Por la propia naturaleza de proyecto de código abierto, muchas implementaciones las han desarrollado parte de su comunidad de desarrollo, por ejemplo el agente de *XCTest*.

Como podemos ver en la ilustración 17, el servicio consta de una serie pestañas donde se encuentran las distintas secciones del sitio web. La primera de ellas muestra un panel de control general de la cuenta. La segunda, es una pestaña donde se almacenan todos los paneles de control que se hayan creado, además es en esta pestaña donde se crean los paneles de control. Hay una pestaña para buscar y filtrar entre todas las pruebas. Otra donde administrar los usuarios que pueden acceder al servicio. Según el uso que haga se le puede dar un rol u otro, limitando de esta forma los permisos de un usuario. Finalmente cuenta con una pestaña de configuración del servicio en general.

La pestaña con el icono del cohete muestra los *launches* o las ejecuciones, donde se puede ver todas las ejecuciones de las pruebas en una lista. Aquí se muestran distintos datos sobre ellas como pueden ser el nombre, quién las lanzó, una serie de etiquetas, la plataforma en las que se ejecutaron, el tiempo de ejecución, hace cuánto se inició la ejecución, el número de pruebas que se han ejecutado (TTL), cuántas han dado un resultado satisfactorio (PS), cuántas dado han dado un resultado fallido (FL), cuántas no se han ejecutado (SKP). Por último, define cuatro campos, en los que se hace un recuento de los fallos que han sido *bugs* del producto, cuántos han sido *bugs* de la automatización, cuántos han sido un fallo del sistema y cuántos habría que investigar para poder resolver.

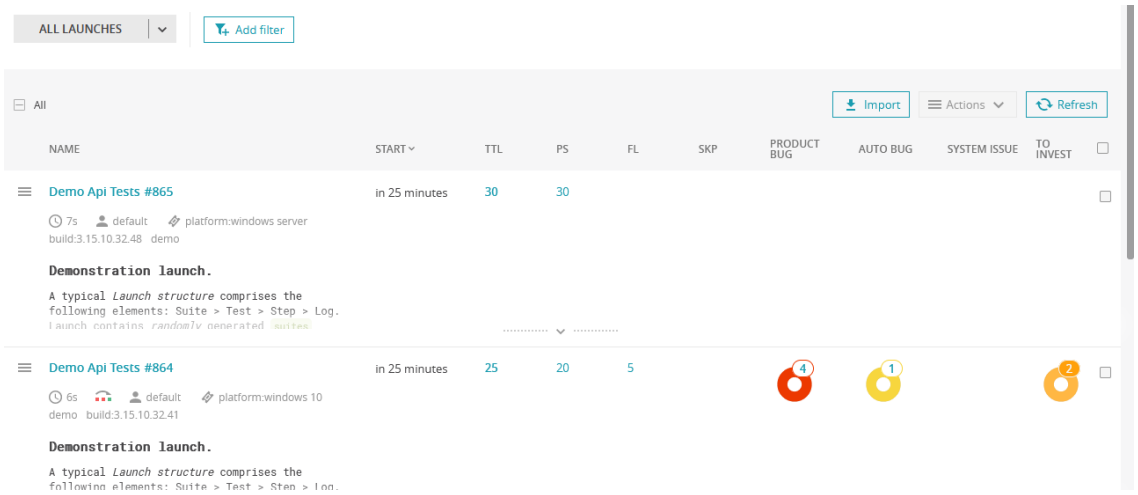


Ilustración 17: Ejecuciones dentro del ReportPortal

Seleccionando los distintos campos se logra que se amplíe más la información, por ejemplo, haciendo clic en el campo TLL de una de las ejecuciones se podrá ver una lista de las pruebas que han pasado correctamente.

Resulta más interesante acceder a un *launch* o ejecución. Este mostrará una pantalla donde se pueden ver todos los conjuntos de pruebas con la misma jerarquía que se encuentran en el proyecto. Dentro de esta vista se repiten los campos usados en el paso anterior, pero ahora viendo la lista de *test cases* que se puede ver en la ilustración 18. Además, es posible editar elementos para añadir una descripción, añadir etiquetas o borrarlo. En la lista de pruebas se puede buscar y filtrar por distintos atributos o requisitos. También existen distintas vistas además de la de lista, por ejemplo, puede verse el registro de la ejecución del *TestCase*. Lo más útil es una vista donde se puede ver un historial de las ejecuciones del *TestCase* que muestra a lo largo de las ejecuciones qué pruebas han pasado, cuáles no y qué tipo de errores tienen. Se puede pasar el ratón por encima para que dé aún más información de la ejecución de esa prueba en concreto y si se le hace clic se podrá acceder a ella.

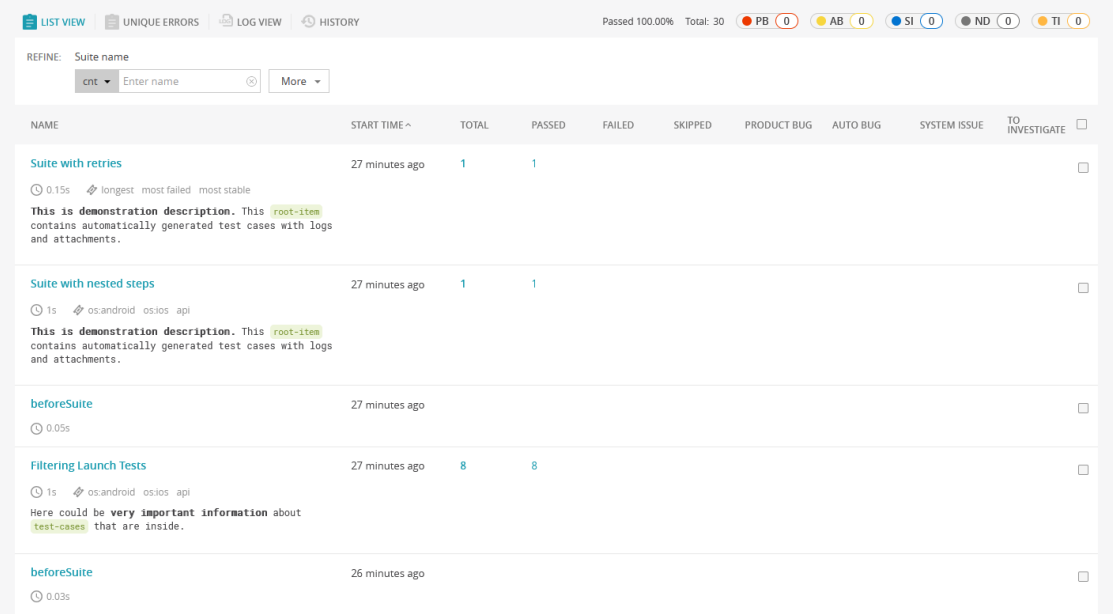


Ilustración 18: Test Cases

Así mismo, al acceder a uno de estos *test cases* se podrán ver las pruebas, dónde se muestra el nombre de la prueba, el tiempo de ejecución, una serie de etiquetas, una descripción, el estado de la prueba, cuándo comenzó y un campo donde pondría el tipo de error si hubiera fallado la prueba. Cuentan con las mismas vistas que las pantallas anteriores, una para poder ver el registro y un histórico de las pruebas en ejecuciones anteriores.

Finalmente, dentro de las pruebas se encuentra en la parte de arriba una línea cronológica donde se sitúan los resultados anteriores de esta prueba en otras ejecuciones. En la parte central se encuentran los registros de la prueba, una pestaña para ver archivos añadidos, una de detalles de la prueba, una historia de acciones y una pestaña con el registro de la pila.

7.1.3. Análisis de la integración con *ReportPortal*

Durante esta fase de estudio previo al desarrollo del proyecto, se ha visto y aprendido cómo funcionan las distintas herramientas que se va a utilizar. Entre ellas, *XCTest* y *ReportPortal*. Ambas son de vital importancia para el desarrollo de este proyecto. Cuando se habló de los agentes de *ReportPortal,* se comentó cómo algunos habían sido creados y mantenidos por los desarrolladores y que *XCTest* no era uno de ellos, si no uno que había sido desarrollado por algún miembro de la comunidad de código abierto.

Al abrir el repositorio de este agente se encuentra el proyecto y la documentación. La única documentación disponible es que, para poder instalarlo, se necesita usar *CocoaPods*, y qué variables se necesitan añadir para que se pueda conectar con *ReportPortal*.

Antes de seguir, hay qué explicar que es *CocaPods*. Es un administrador de dependencias desarrollado en *Ruby* y creado para funcionar en proyectos que usarán *Objetive-c* y más adelante *Swift*. Tiene una gigantesca variedad de librerías y es ampliamente usado por toda la comunidad de desarrolladores *iOS*; se asemeja a lo que es *NPM* para desarrolladores de JavaScript. Así mismo hay que comentar que también existe integrado en *Xcode* un administrador de dependencias basado en *Swift*, el *Swift Package Manager*. Este gestor es mucho más joven, tiene mucho menos soporte por parte de los desarrolladores y cuenta con bastante menos librerías disponibles por el momento.

Tras leer la escueta documentación del agente se intentó añadir esta librería o *pod* al proyecto usando los comandos proporcionados por la documentación del repositorio. No hay problemas a la hora de instalar la librería. Se añadieron las variables necesarias para que fuera capaz de conectarse satisfactoriamente al *ReportPortal*, pero no funcionaba. Seguramente lo que ha pasado es que el desarrollador que creó este agente no ha podido actualizarlo. Esta es una de las desventajas de este tipo de proyectos.

No encontrando solución a este problema, lo que se planteó fue arreglar el agente y además de esto, mejorarlo, pero primero se tendría que entender cómo funciona en general las interconexiones entre *XCTest* y *ReportPortal*.

Los agentes se comunican con *ReportPortal* mediante una *API,* así que se necesita un servicio que sea capaz de enviar peticiones y recibir repuestas *http*. Principalmente, el agente va enviando peticiones y esperando respuesta a los siguientes eventos:

N.º del evento	Nombre del evento	Envía	Recibe
1	Inicio ejecución	Nombre de la ejecución	Identificador de la ejecución
2	Inicio *test case*	Nombre del *test case* e identificador de la ejecución.	Identificador del *test case*
3	Inicio de la prueba	Información de la prueba, id de la ejecución e identificador del *test case*	Identificador de la prueba
4	Envió registro	Registro de la prueba	Identificador del registro

5	Acaba la prueba	Información de la finalización de la prueba, resultado e identificador de la prueba	Confirmación
6	Acaba el *test case*	Información sobre la finalización del *test case* e identificador de este	Confirmación
7	Acaba la ejecución	Información sobre la finalización de la ejecución e identificador de esta	Confirmación

Tabla 3: Eventos para registrar pruebas en ReportPortal

Se ha de destacar que los pasos del dos al seis se repiten según la cantidad de pruebas que tenga ese *test case* y según la cantidad de *test cases* que tenga la ejecución. Cada una de estas peticiones tiene su formato de campos y atributos con los que tiene ser enviada y su respectiva respuesta, si todo se envía correctamente.

Además, se puede ver cómo en el repositorio se dispone de una carpeta llamada *EndPoints* donde se encuentran estructuras de archivos que definen los contenidos de las distintas peticiones que enviamos. Por ejemplo, para enviar el resultado de una prueba se necesita el método *http* con el *endpoint* a usar, una ruta relativa y una matriz de parámetros donde se encuentra el identificador del objeto, el mensaje y la hora en la que se envió.

También cuenta con una carpeta llamada *Entities*, en la que se encuentran distintas estructuras de datos que se usan en el agente. Así mismo, la última carpeta que se encuentra es la carpeta de *Utilities,* donde se aloja una clase con un método capaz de llamar a un *EndPoint* por *http*. Una clase que ayudará con las etiquetas, aunque solo asigna etiquetas a la ejecución, algo que no es ideal. Otro elemento que se encuentra en el repositorio es la clase que ayuda a gestionar la autorización en las llamadas *http*. Otra clase para formatear fechas conforme a los requisitos de la *API* de *ReportPortal*. Por último, un método que devuelve la versión de *iOS* y el dispositivo que se está usando para poder añadir esa información a la ejecución.

Por último, en la raíz del repositorio se encuentran las dos clases principales donde se aloja casi toda la lógica del agente, un archivo de configuración y varios enumerados que definen variables posibles para *LaunchMode*, *TestStatus* y *TestType*.

Por otro lado, de *XCTest* se analizará cómo el *framework* de *Apple* ejecuta las pruebas para poder enviar las peticiones necesarias a *ReportPortal*. Al investigar en el proyecto, se puede descubrir que importa a *XCTest* en dos archivos, en uno de ellos solo llama para usar

XCTestCase como un objeto y en el otro usa un elemento que se desconocía, pero parece prometedor, *XCTestObservation*.

Al investigar sobre *XCTestObservation* no se encuentra mucha información, tan solo la documentación de *Apple*. Se trata de un protocolo (un protocolo es un elemento muy parecido a las interfaces en otros lenguajes de programación). Define los distintos métodos que se irán ejecutando según empiecen y acaben los distintos eventos que ocurren durante la fase de pruebas. Estas funciones son las siguientes:

- *testBundleWillStart*
- *testSuiteWillStart*
- *testCaseWillStart*
- *testCase*
- *testCase* (cuando una prueba se espera que falle)
- *testCaseDidFinish*
- *testSuite*
- *testSuite* (cuando un conjunto de pruebas se espera que falle)
- *testSuiteDidFinish*
- *testBundleDidFinish*

Todas estas funciones son fáciles de comprender gracias a sus nombres descriptivos, que ayudan a saber a qué parte del ciclo de ejecución pertenece cada una de ellas.

Con esta información ya se entiende mucho mejor cómo funciona este agente y cómo se conecta con *ReportPortal*. También cómo se puede conseguir la información que se quiere enviar al servicio con *XCTest*. Todo este conocimiento será de gran utilidad a la hora de mejorar la implementación y poder añadir nuevas funciones.

7.2. Desarrollo

Durante esta sección del documento se explicarán los pasos necesarios para poder implementar el servicio y cumplir satisfactoriamente con los objetivos de este proyecto. Se empezará con la implementación de la canalización dentro del repositorio. El siguiente paso será la implementación de la fase de compilación y pruebas. Más tarde, el despliegue de *ReportPortal*. Luego, se le dedicará tiempo a mejorar la actual implementación del agente de *XCTest*, se comprobará la conexión de las pruebas con *ReportPortal* y, por último, la creación del informe.

7.2.1. Implementación de la canalización al repositorio

Lo primero que se desarrollará será la canalización, para ello se accede a la consola de *Amazon Web Services* y dentro se buscará el servicio de *CodePipeline*. La creación de una canalización consta de cinco pasos que hay que completar: configuración de la canalización, agregar la etapa de origen, agregar la etapa de compilación, agregar la etapa de implementación y revisar y confirmar todas estas configuraciones.

En la primera etapa de la creación de una canalización nueva, mostrará una pantalla de configuración en la que se tendrán que rellenar distintos campos: nombre de la canalización y qué rol de servicio va a usar, si uno ya creado o uno nuevo. Además, existe una configuración avanzada que permite decidir dónde se van a guardar los artefactos generados por la canalización dentro de *S3* y quién va a gestionar la clave de cifrado, si *AWS* o el cliente. Se dejan todos los campos en su configuración por defecto menos obviamente el nombre, donde se usará, *TFG_iOS*.

En el segundo paso de este proceso se configura la etapa de origen de la canalización. Como se puede ver en la ilustración 19, lo primero que se tiene que escoger es el proveedor de esta acción, dentro la amplia lista, se escogerá *BitBucket*. Ahora aparecerán más campos que se tienen que rellenar. En este caso pide una conexión con el repositorio. Se puede escoger una ya existente o crear una nueva conexión.

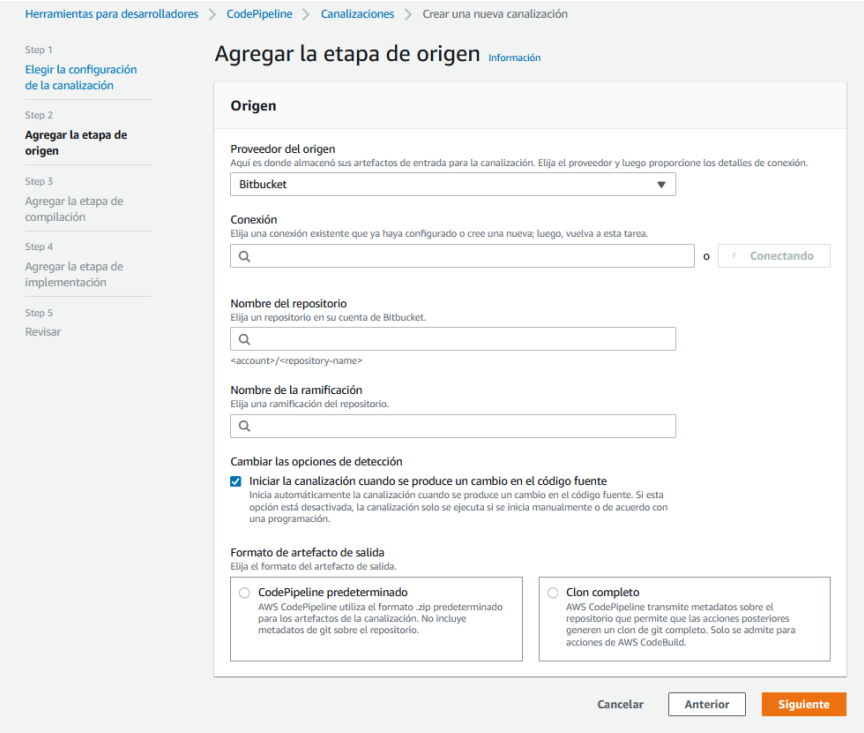


Ilustración 19: Agregar etapa de origen

El proceso de generar una nueva conexión es bastante sencillo. Al apretar el botón muestra una nueva pestaña, en esta se configurará la conexión. Es necesario darle un nombre y luego generar una *BitBucket App,* que se genera accediendo a una cuenta de *BitBucket* y aceptando darle permisos a *AWS CodeStar* para poder acceder a repositorios concretos. Con estos pasos ya se crearía una conexión nueva.

Una vez asignada la conexión, solo falta elegir el nombre del repositorio que se va a usar y el nombre de la rama. Se deberá activar la opción de iniciar la canalización cuando se produce un cambio en el código fuente, para que esta fase de origen sea el desencadenante de toda la canalización.

La última decisión por tomar es escoger el formato de los artefactos de salida. Por un lado, se tiene el *CodePipeline* predeterminado donde se guarda como un zip y no incluye metadatos del repositorio. Por otro lado existe el clon completo donde también se empaquetan todos los metadatos, pero no comprime el proyecto. Este limita a usar sí o sí el proveedor de *AWS CodeBuild* en lugar de *Jenkins* en la siguiente fase de construcción, así que se usará el predeterminado.

Ahora se explicará la tercera fase de la configuración, agregar la etapa de compilación que se muestra en la ilustración 20. Aunque la implementación de la fase de compilación y pruebas es otro apartado del desarrollo, ahora mismo se necesita asignar mínimo dos etapas a la canalización para poder crearla. En esta fase se encuentra una pantalla donde primero se tiene que seleccionar el proveedor, el cual será *Jenkins Personalizado*. Después de esto pregunta por un nombre para el proyecto de *Jenkins*. El segundo campo por rellenar es el nombre del proyecto dentro de *Jenkins*. El valor que se use será el mismo nombre que se usará para crearlo dentro de *Jenkins*. Una vez rellenado todo se pasa a la siguiente fase.

Ilustración 20: Agregar etapa de compilación

La cuarta fase de la configuración, agregar la etapa de implementación, se puede omitir, ya que no hace falta en este proyecto. Además, el proyecto ya cuenta con el mínimo de dos etapas dentro de la canalización. Se pregunta por confirmación para omitirla y listo.

La última fase de la configuración no es nada más que una fase de revisión. Se puede ver la configuración elegida para poder realizar algún cambio si lo consideramos oportuno o por si se encuentra algún error. Tras confirmar toda la configuración, se crea la canalización y ya se deberá poder ver cómo la canalización comienza y ejecuta sus dos etapas: importa el último evento del repositorio e intenta compilar.

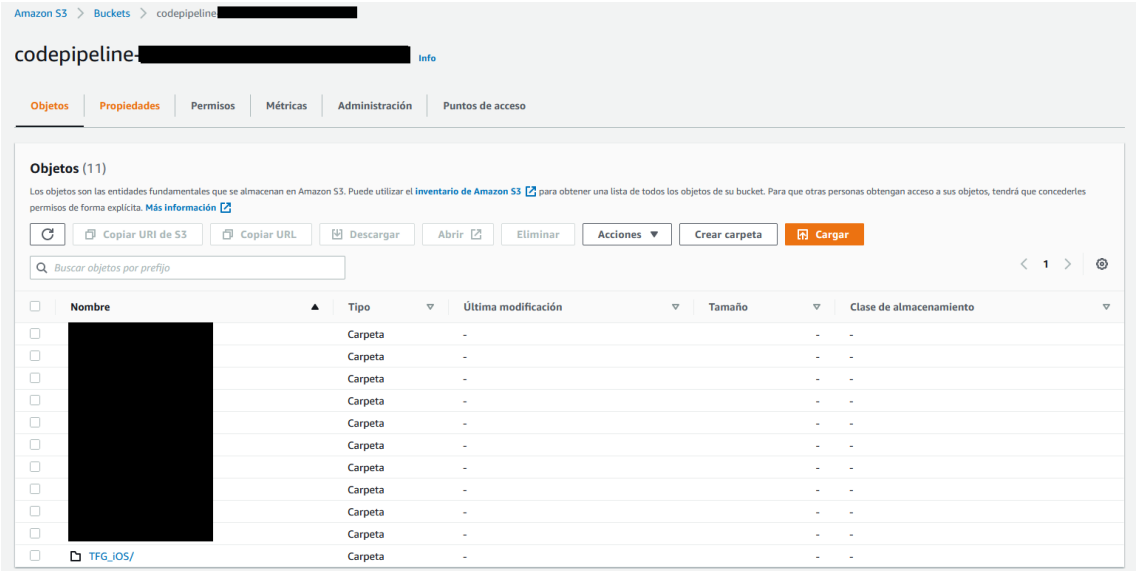


Ilustración 21: Carpeta de artefactos

Efectivamente se puede ver cómo adquiere del repositorio el último evento de la rama, en este caso era una fusión desde otra rama. Lo empaqueta en un archivo *zip* que se encuentra en el *bucket* de *S3* como se puede ver en la ilustración 21. Este artefacto que genera es mandado a *Jenkins* en la segunda etapa de la canalización y genera un error ahora mismo ya que no está configurado todavía.

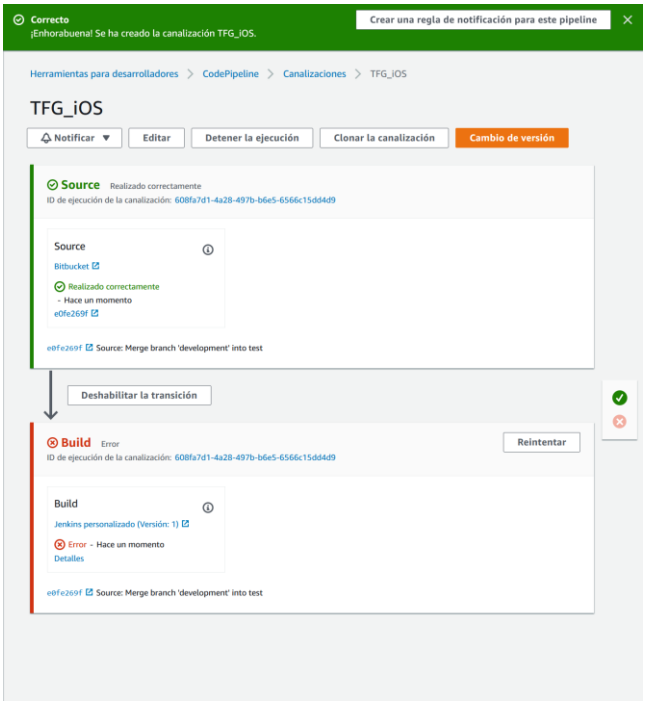


Ilustración 22: Canalización

Se explicará cómo funciona la interfaz de *CodePipeline* que se ve en la ilustración 22. Dentro de las canalizaciones se encuentra una lista de etapas. Dónde la primera es la que lanza la propia canalización según el evento deseado. Se van ejecutando en orden descendente una por una y así mismo sus acciones siguen el mismo orden de ejecución, pero también se pueden ejecutar en paralelo. Se pueden añadir notificaciones para enviar alertas según los distintos eventos de la canalización. Se puede editar la canalización para poder añadir etapas, borrarlas y editarlas y lo mismo ocurre con las acciones, todo con una interfaz gráfica bastante intuitiva. Así mismo, se puede clonar la canalización y también se puede obligar un cambio de versión. Esto lo que hace es que se ejecute la canalización de principio a fin sin tener que esperar por el evento detonante en la etapa *source*.

7.2.2. Implementación de la fase de compilación y pruebas

Tras conseguir implementar la fase de canalización del repositorio, se va a empezar con la fase de implementación de la compilación y pruebas. Como este proyecto se va a orientar al desarrollo de *iOS,* necesita ejecutar la compilación y las pruebas usando *MacOS,* ya que es la

única forma de hacerlo. Este fue uno de los motivos por el cual se decidió usar los servicios de *Amazon Web Services*: es la una de las pocas plataformas de computación en la nube que cuenta con máquinas capaces de ejecutar *MacOS*. Dentro de esta máquina se instalará *Jenkins* para poder automatizar todos estos procesos.

El primer paso es el despliegue de una instancia *EC2* con *MacOS*. Para ello se abre la consola de *AWS* y se buscará *EC2*. Esto muestra un panel de control como el que se muestra en la ilustración 23con una gran cantidad de información. Aquí se puede ver todo tipo de información relacionada con *EC2*: los recursos, atributos de la cuenta, información adicional, estado del servicio, lanzar la instancia, eventos programados, migrar un servidor e información adicional.

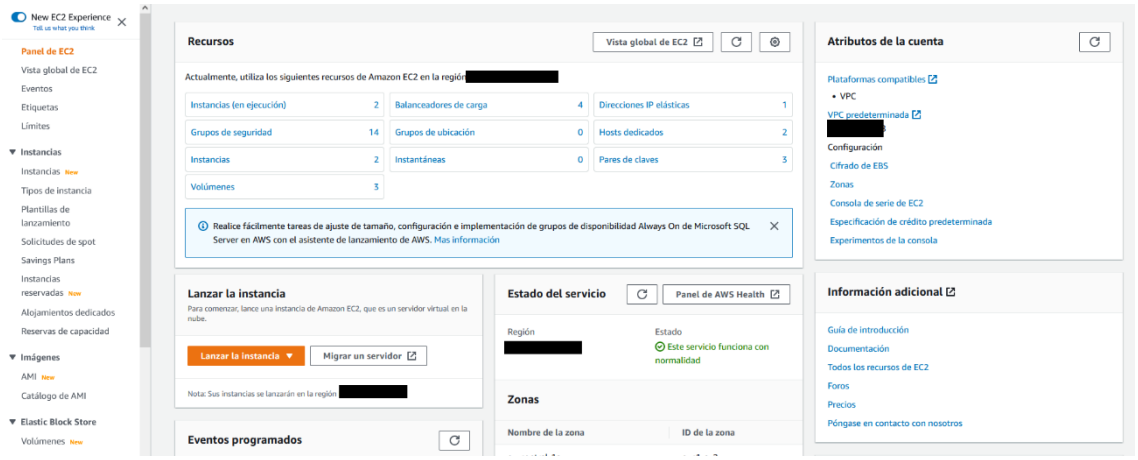


Ilustración 23: Panel de control EC2

En el panel de la derecha se encuentran distintas acciones que se pueden hacer agrupadas en los siguientes campos: Instancias, Imágenes, *Elastic Block Store*, Red y Seguridad, Equilibrio de carga y *Auto Scaling*. Lo que se debe hacer es lanzar una instancia, así que simplemente se aprieta el botón. Esto muestra la vista donde se debe configurar la instancia.

Primero se le asigna un nombre a la instancia teniendo la posibilidad de asignarle etiquetas adicionales en forma de clave valor. El siguiente apartado a configurar es la selección del sistema operativo o la imagen, como ya se ha comentado durante este documento se buscará *MacOS*.

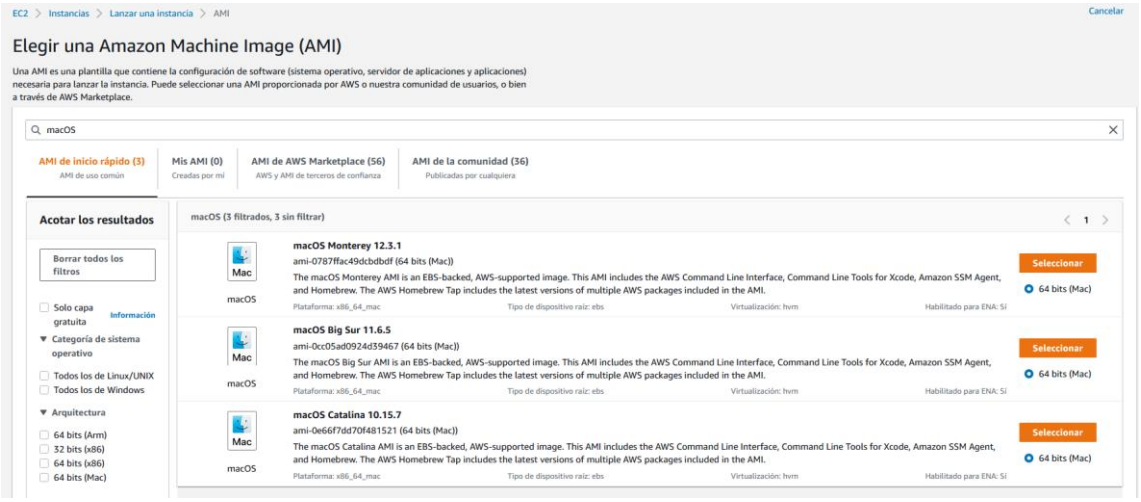


Ilustración 24: Vista selección imagen

Como se puede apreciar en la ilustración 24, de la búsqueda, se encuentran, en la pestaña de inicio rápido, tres versiones distintas del sistema operativo, estas son las oficiales de *Amazon Web Services*. De igual modo, se pueden encontrar dentro de un *Marketplace* una serie de imágenes desarrolladas por terceros de confianza. Igualmente pasa con la última pestaña en donde se pueden encontrar imágenes publicadas por la comunidad. Para este proyecto se usará la versión más reciente de las de inicio rápido, MacOS Monterey 12.3.1.

Una vez seleccionada se muestra la pantalla anterior donde se tiene que elegir ahora el tipo de instancia (básicamente el hardware que tendrá la instancia). Para *MacOS* solamente existe un tipo de instancia, la *mac1.metal*. La instancia consta de doce núcleos virtuales, se ejecuta bajo la arquitectura *x86_64*, usa 32 *Gb* de memoria y tiene una interfaz de red con un ancho de banda de diez *Gigabit*. Se seleccionará ese tipo de instancia para proseguir.

Después se le debe asignar un par de claves para el inicio de sesión. Se pueden seleccionar unas que ya estén disponibles o crearlas. Se crearán unas para este proyecto. Para ello se selecciona el botón de crear. Aparecerá una ventana como la que se aprecia en la ilustración 25, en la que se deberá asignarles un nombre, elegir la encriptación y el formato del archivo de la clave privada. Como encriptación se usará *RSA* y como formato de la clave privada se usará *.pem*. Una vez todo rellenado y creado se debería descargar el archivo con la clave privada.

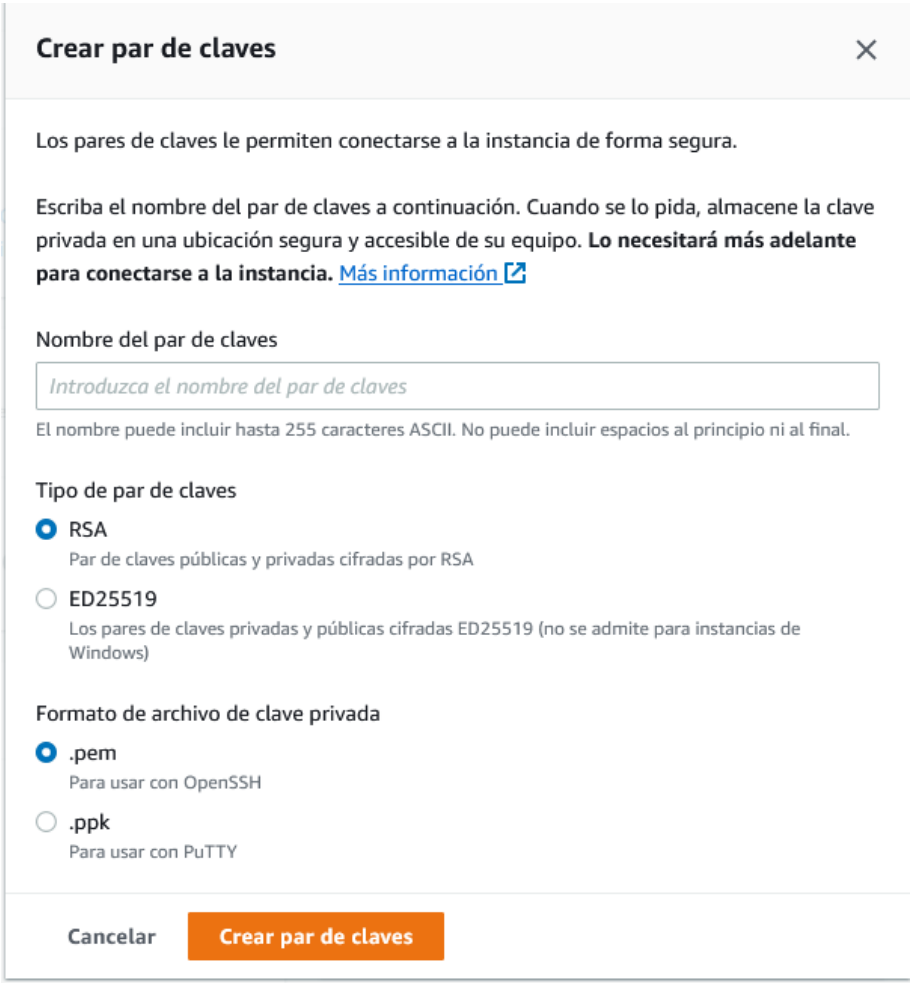


Ilustración 25: Creación claves

Dentro de la configuración de red que se puede ver en la ilustración 26, se tiene que asignar un *VPC* para que se pueda conectar a la red y a distintas máquinas que se encuentren bajo la *VPC*. Además de esto, también se le puede asignar una subred dentro de la *VPC*, asignarle la dirección *IP* o que se genere de forma automática. Habría que decir también que con la creación de la instancia se genera un *Firewall*, aunque también se puede asignar uno que ya esté creado.

El *Firewall* necesita un nombre, una descripción y las reglas de seguridad. Dentro de estas se puede aplicar restricciones a la entrada de conexiones como, por ejemplo, la regla por defecto, que consiste en permitir conexiones tipo *SSH*, usando el protocolo *TCP* y en el puerto 22, con un origen desde cualquier lugar y con cualquier *IP*. Se necesita añadir las siguientes entradas de conexiones, la primera es para poder acceder mediante *VNC* a la instancia. Esta tecnología usa los puertos 5900, 5800, 5500; también se añadirá el puerto 8080 que es el que utiliza *Jenkins* para poder usar la interfaz gráfica.

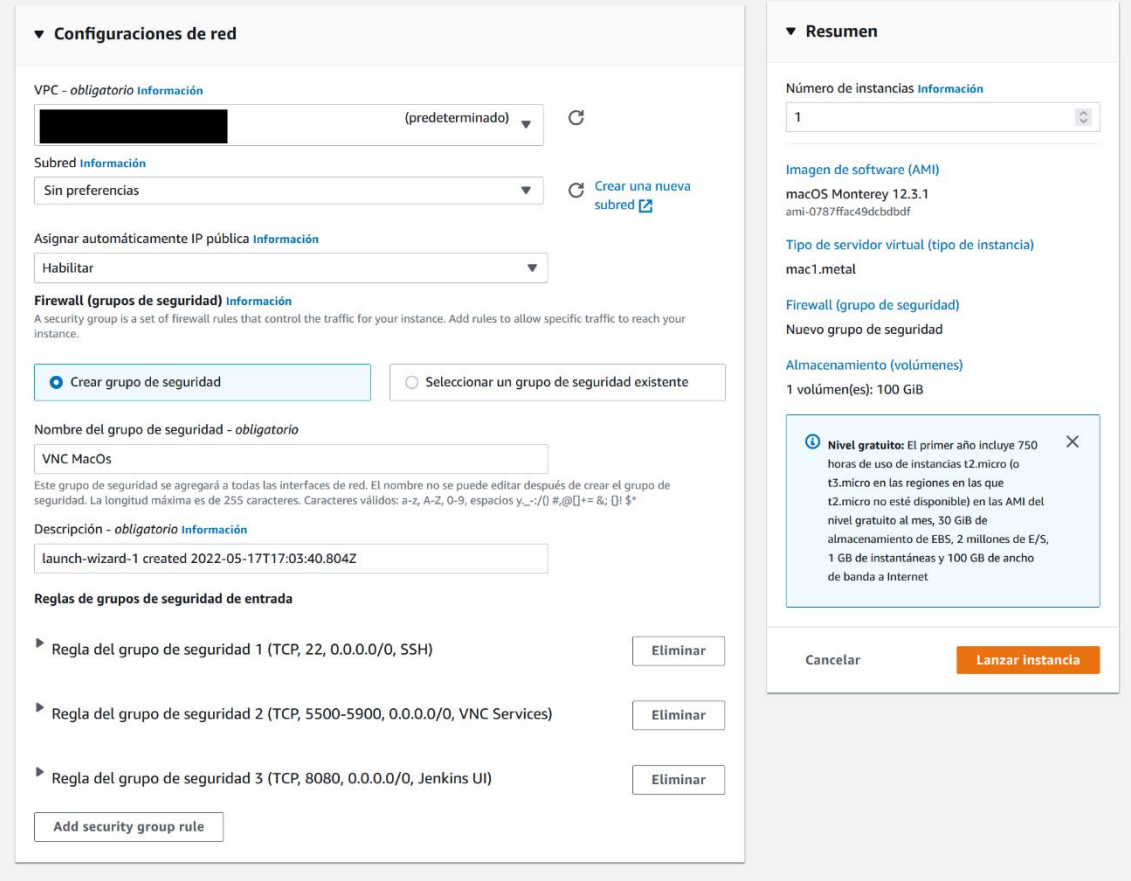


Ilustración 26: Configuración de red

Hay que mencionar que, aunque se ha configurado para que los servicios sean accesibles desde cualquier *IP,* esto se hace por no mostrar una *IP* privada ni la de los equipos de la empresa. Esto es una mala práctica para la seguridad de las instancias, y *Amazon Web Services* alerta sobre ello.

Por otro lado, se le puede añadir almacenamiento a la instancia. Se puede seleccionar varios tipos de almacenamiento: SSD, SSD orientado a IOPS, HDD y discos magnéticos. También se puede seleccionar el tamaño para el volumen raíz. Además, es posible añadir más almacenamiento definiendo los mismos campos que el volumen raíz.

Por último, también existen una serie de configuraciones avanzadas entre las que cabe destacar: GPU elástica (brinda la capacidad de añadir GPU a la instancia), recuperación automática de las instancias, comportamiento al cerrar (permite definir un comportamiento cuando se cierra la máquina) y monitorización detallada de *CloudWatch*. Aunque sean configuraciones que no se van a usar, siempre está bien tenerlas en cuenta, ya que pueden llegar a ser útiles.

Finalmente se revisa la configuración por si se ha cometido algún error y después se lanza la instancia. En el momento que se haga esto, se abrirá a una ventana de carga y luego la vista general de las instancias como se observa en la xilustración 27. Aquí se puede ver la instancia que se encuentra iniciando, habrá que esperar un poco hasta que se encuentre en estado de ejecución.

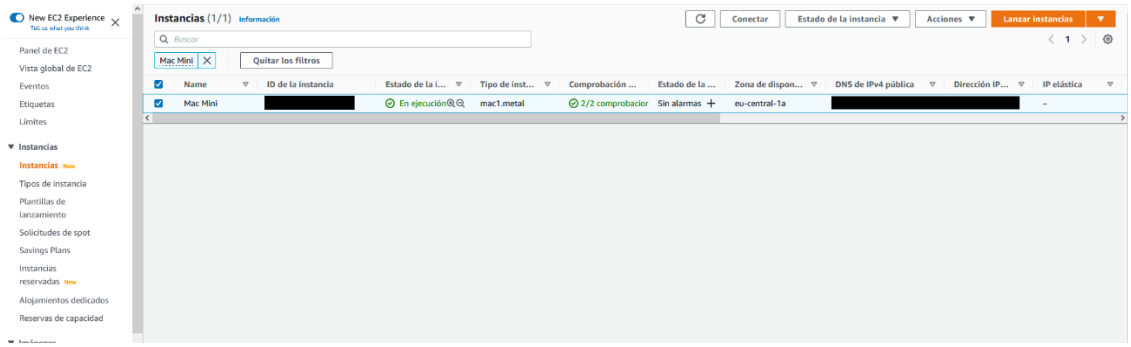


Ilustración 27: Instancia lanzada y en ejecución

Una vez la máquina esta iniciada ya se puede conectar a ella. Para eso lo primero será hacer clic en el identificador de la instancia. Se mostrará un panel que muestra toda clase de información. En la parte principal, un resumen de las características de la máquina que aporta información como la dirección *IP* de la instancia pública y privada, el *VPC*, el identificador de la subred en la que se encuentra y el tipo de instancia. Dentro del mismo panel más abajo se encuentra más información separada por secciones.

En relación con el proyecto, el siguiente paso sería la instalación de *Jenkins* dentro de la instancia que acabamos de generar, pero antes de poder instalar el servicio se necesita poder conectarse a la instancia. En la pantalla donde se pueden ver las diferentes características de la instancia, en la esquina superior derecha, se encuentran un conjunto de botones, entre ellos el de conectar.

Como resultado aparece una pantalla que permite ver las distintas formas de conexión con la instancia. Se puede conectar a través de una plataforma que proporciona *Amazon Web Services*, por un administrador de sesiones que se tendrá que instalar y configurar en la instancia, por una consola en serie la cual no está disponible para *MacOS* y la que se usará para este proyecto, que será a través de un cliente *SSH*.

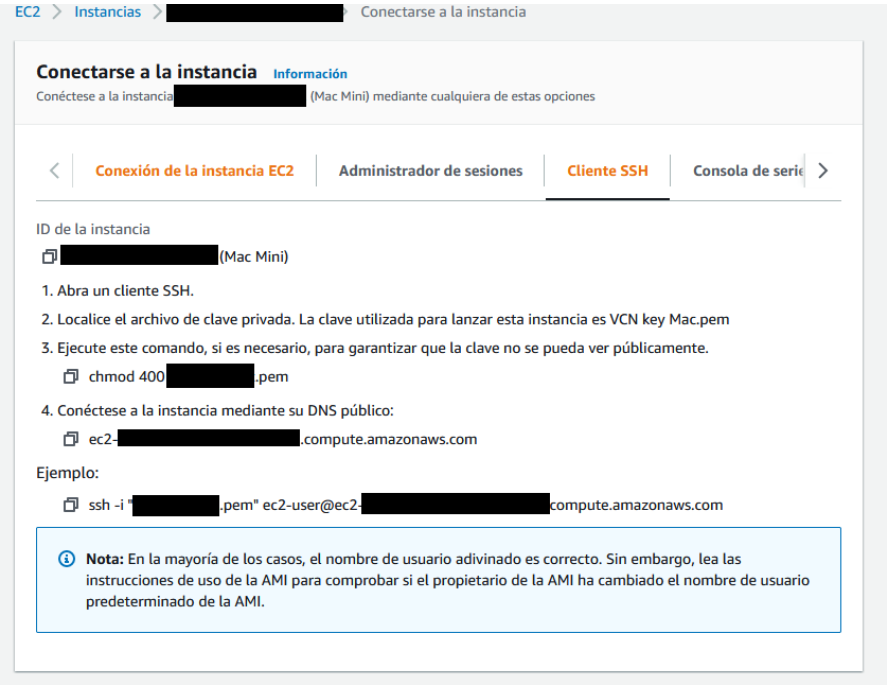


Ilustración 28: Como conectarnos por SSH

Al acceder a la pestaña se puede ver una pequeña guía de cómo se conecta a la instancia mediante *SSH*. Primero se busca el directorio donde se ha guardado la clave privada, que se generó cuando se creó la instancia. Luego se le cambian los permisos al archivo para poder ser ejecutado por cualquier usuario y se accede las propiedades de este. Dentro de la pestaña de seguridad, se accede a editar, seleccionamos el grupo y en la parte de abajo cambiamos los permisos. Después es necesario situarse en la carpeta adecuada usando la consola de *Windows* y se ejecuta el comando que se indica en el ejemplo de la ilustración 28. Tras ejecutarlo nos aparecerá lo que se puede ver en la ilustración 29 que significa que ya estamos conectados. Por motivos de seguridad no se muestra la dirección, pero la forman la dirección *IP* de la instancia unida con la región donde se aloja.

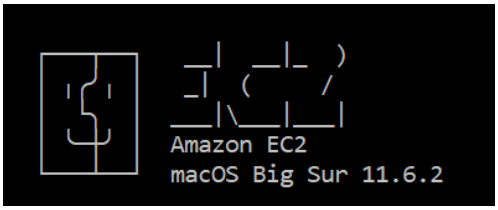


Ilustración 29: Conectado por SSH

Tras ejecutar el comando se conectará a la instancia, pudiendo instalar todo lo que se necesita. Esto no es ideal ya que *Jenkins* usa una interfaz gráfica para ser administrado. Para solventar eso, la conexión a la instancia será vía *VNC* (software libre para poder conectarse a un ordenador por la red y poder usarlo). Lo único que se necesita hacer es asignarle una

contraseña al usuario por defecto *EC2-user*. Simplemente se ejecuta el comando "`sudo passwd EC2-user`" y se le cambia la contraseña del usuario por defecto.

A partir de aquí se usará una herramienta llamada *TightVNC* para conectarnos a la instancia. La herramienta solamente necesita la dirección de la instancia. Una vez que compruebe que existe pedirá un usuario, la contraseña y se conectará como se muestra en la ilustración 30.

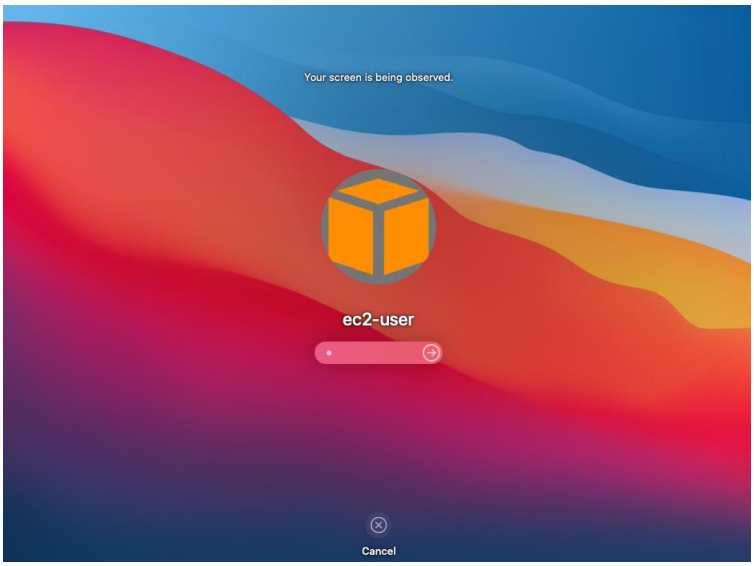


Ilustración 30: Conectado por VNC

Tras iniciar sesión ya se puede empezar a instalar *Jenkins*. Pero antes se necesita instalar *Homebrew,* un administrador de paquetes para *MacOS* y *Linux*. Simplemente se tendrá que lanzar el siguiente comando en la consola de la instancia: "`/bin/bash -c "$(curl fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`" y esperar que el proceso de instalación se complete como se ve en la siguiente ilustración.

```
==> Installation successful!

==> Homebrew has enabled anonymous aggregate formulae and cask analytics.
Read the analytics documentation (and how to opt-out) here:
  https://docs.brew.sh/Analytics
No analytics data has been sent yet (nor will any be during this install run).

==> Homebrew is run entirely by unpaid volunteers. Please consider donating:
  https://github.com/Homebrew/brew#donations

==> Next steps:
- Run brew help to get started
- Further documentation:
    https://docs.brew.sh
```

Ilustración 31: Instalación Homebrew

Una vez completados los pasos anteriores, se ejecuta el comando de instalación de *Jenkins* "`brew install Jenkins-lts`", y como en el caso anterior habrá que esperar a que concluya su instalación. Después de este proceso se lanzará *Jenkins* con el comando que figura en el registro de la instalación "`brew services start Jenkins-lts`". Este proceso podría tardar.

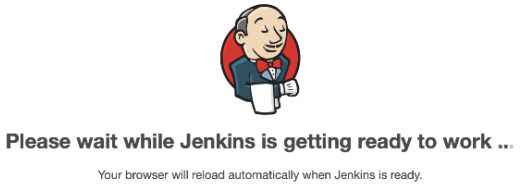


Ilustración 32: instalación Jenkins

Como resultado se debe tener *Jenkins* corriendo de fondo y pudiendo acceder a él con la dirección *localhost:8080* (Ilustración 32). A continuación, se puede ver una pantalla para desbloquear *Jenkins* como se puede apreciar en la ilustración 33. Esto se hace como medida de seguridad para que solo administradores sean capaces de configurarlo. Lo que se debe de hacer para desbloquearlo es acceder al archivo de la dirección para poder ver la contraseña, usando el comando "`cat`" con la dirección se debería mostrar en la consola la contraseña, se copia y pega y así es posible seguir avanzando.

Ilustración 33: Desbloquear Jenkins

La siguiente página que se mostrará es para poder personalizar la instalación de *Jenkins*. Se debe elegir entre las dos opciones, instalar los *plugin* sugeridos o escoger qué *plugins* serán instalados. Para este proyecto se elegirá la instalación con los *plugins* sugeridos y se pasará a una fase de instalación. El último paso de la instalación será crear el primer usuario administrador y ya estaría instalado. Finalmente se muestra la pantalla principal de *Jenkins* donde se pueden administrar todos los trabajos que se hayan creado.

Lo siguiente que se debe de hacer es la configuración de *Jenkins* para que sea capaz de integrarse con *CodePipeline* y que se ejecute en la fase de *CodeBuild*. Se accederá a la configuración de *Jenkins,* que se encuentra en el panel de control, dentro de la barra lateral a la derecha. Dentro de esta se entra a la administración de los *plugin*. Se buscarán los siguientes *plugins*, se marcan todos y se instalan:

- *AWS CodeBuild Plugin*
- *AWS Web Services SDK :: All*
- *Amazon EC2*
- *AWS CodePipeline*
- *Email Extension*

Una vez instalado ya se puede proceder a la creación del trabajo que va a ejecutar las pruebas.

Antes de continuar, es imprescindible instalar *Xcode* para poder ejecutar las pruebas. Para ello se necesita iniciar sesión con un *Apple ID* válido en el *Apple Store*, ahí se busca *Xcode* y se instala. También se necesita instalar *Xcode Command Line Tools* con el objetivo de ejecutar *Xcode* desde la línea de comandos. Para hacer esto, simplemente se abre la consola de comandos y se lanza la siguiente orden "`xcode-select -install`". Esto iniciará el proceso de instalación. Una vez acabado ya se tendrán instaladas las herramientas de *Apple* para el desarrollo.

Ahora se cumplen todos los requisitos previos para poder compilar y probar aplicaciones iOS. Lo siguiente será configurar el trabajo que ejecutará las pruebas en *Jenkins*. Lo primero que se hará será acceder a la opción *New Item,* en la barra lateral de la derecha dentro del panel de control.

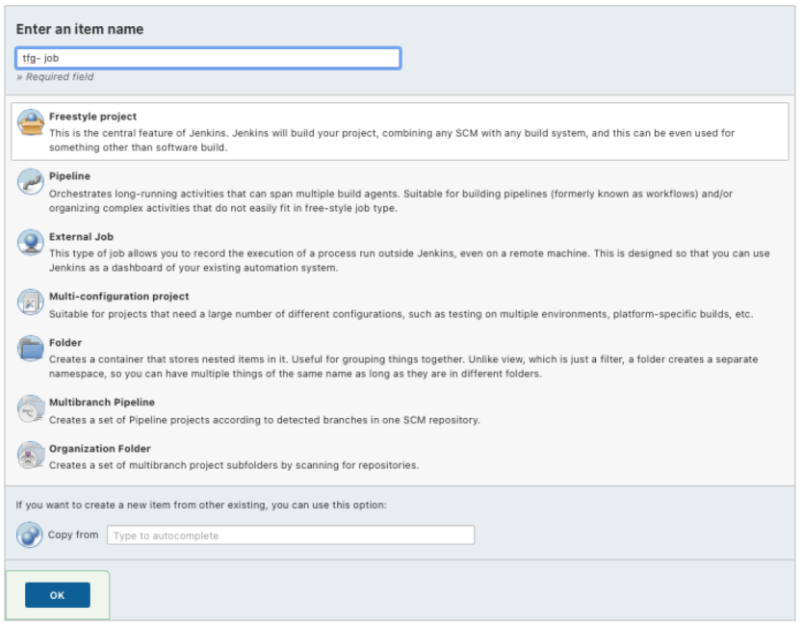


Ilustración 34: Creación de un objeto dentro de Jenkins

Como se puede ver en la anterior ilustración, dentro de la siguiente vista se selecciona el tipo de trabajo. Se escoge un proyecto libre y se le asigna un nombre. En la siguiente pantalla de la configuración se encuentra todo lo que hará este trabajo, *Jenkins* lo divide en seis partes que se tendrán que ir configurando: *General, Source Code Management, Build Triggers, Build Enviroment, Buils* y *Post-Buils Actions*.

La primera parte de la configuración son los ajustes generales, en los cuales se encuentran varias opciones como se ve en la ilustración 35. Las que se usarán para este proyecto son la descripción del proyecto, una opción de descartar compilaciones antiguas y se establecerá una estrategia llamada *Log Rotation*.

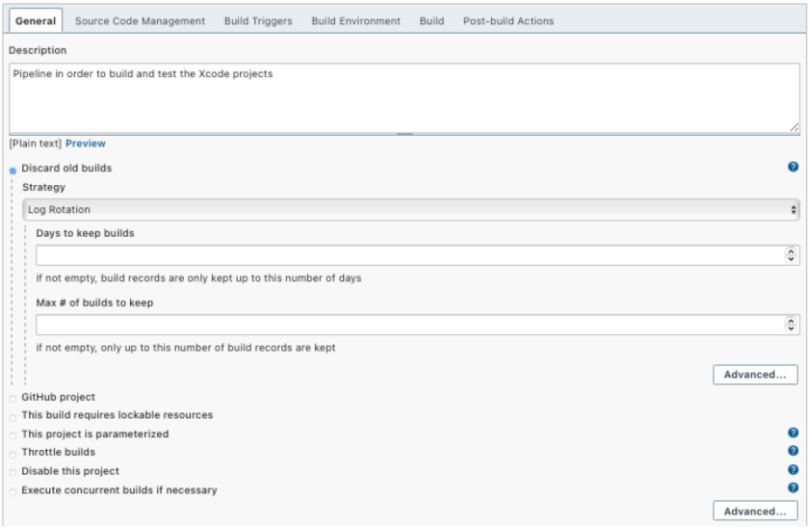


Ilustración 35: Configuración general de un objeto en Jenkins

La segunda parte, es la fase de configuración del proceso del código fuente. De las opciones que aparecen se seleccionará *AWS CodePipeline*, tras esto se desplegarán los ajustes a rellenar. Empezando con la región de *AWS*, donde se tiene que escoger la misma región donde estén alojadas la instancia y la canalización. Lo siguiente que se tendrá que rellenar son las credenciales. Se necesita generar unas credenciales de seguridad para que este servicio sea capaz de conectarse a nuestra canalización. Con este objetivo se vuelve a acceder a la consola de *Amazon Web Services* y luego se buscará *IAM*. Se creará un usuario con permisos dentro de la canalización, así que se selecciona la opción añadir usuario.

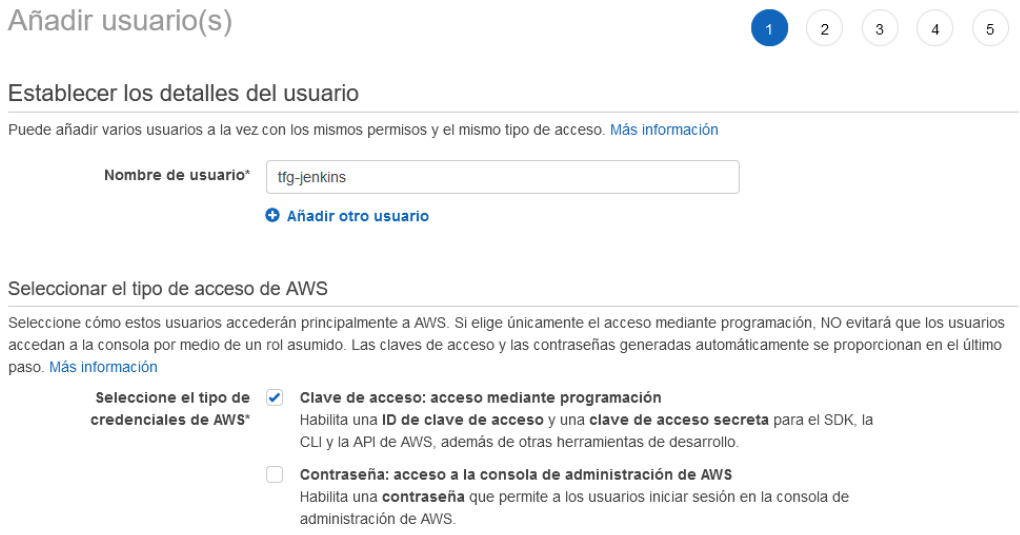


Ilustración 36: Creación de un usuario AWS IAM

Se le asigna un nombre y se marca en el tipo de credenciales la opción clave de acceso como se aprecia en la ilustración 36. En el segundo paso se establecen los permisos, se hace clic en la pestaña asociar directamente las políticas existentes y dentro se busca la política que deseamos aplicarle, en este caso *AWSCodePipelineCustomActionAccess*. El siguiente paso es añadirle etiquetas, que es opcional. En el penúltimo paso solamente se tendrá que revisar la configuración y pulsar en crear usuario. Finalmente, ya estaría creado este usuario y muestra su identificador de clave de acceso y su clave de acceso secreta como se muestra en la siguiente ilustración.

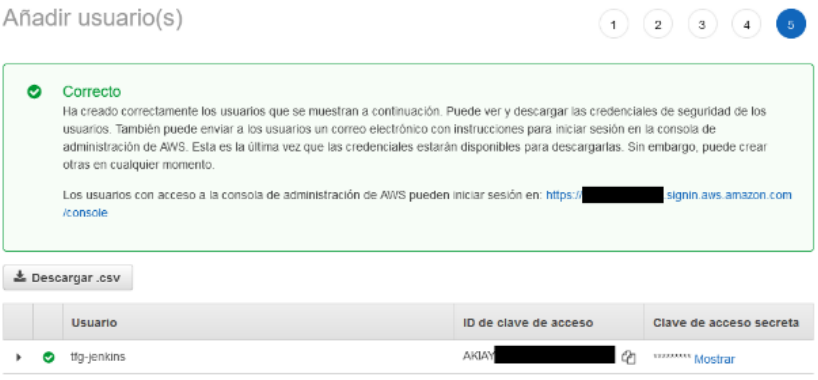


Ilustración 37: Clave publica y privadas

Hay que aclarar que esta es la única vez que puede ver la clave secreta, así que habría que guardarla en un lugar seguro. También se puede desactivar las claves de acceso y generar unas nuevas en el caso de que se haya olvidado o perdido la actual.

Dentro de la configuración de *Jenkins* se usan las credenciales que se acaban de crear. Lo siguiente que se hará es seleccionar el tipo de acción en el *Pipeline CodePipeline Action Type*, la cual sería *Build*. Por último, en este apartado de la configuración solo se tiene que indicar el nombre del proveedor que pusimos en la acción del *CodePipeline* y la versión como se ve en la ilustración 38.

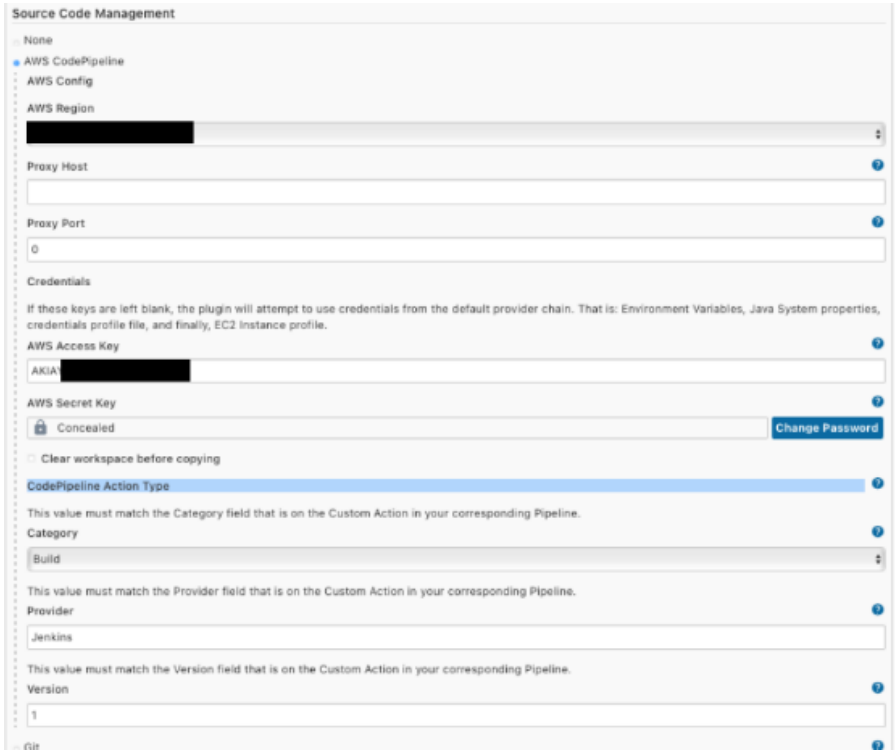


Ilustración 38: Configuración de origen en un objeto en Jenkins

La siguiente fase de configuración del proyecto es *Builds Triggers*. En esta fase se le indica al trabajo la frecuencia con la que debe buscar si existen cambios en el proyecto para poder ejecutarse. Dentro de la opción de *Poll SCM* se escriben cinco asteriscos separados por un espacio para decir que la frecuencia será una vez cada minuto como se aprecia en la siguiente ilustración.

Ilustración 39: Configuración desencadenadores de compilación de un objeto en Jenkins

Otra de las fases de configuración es la de *Build Enviroment*, donde se pueden aplicar una serie de configuraciones en el entorno en el cual se ejecuta la compilación. Dentro de esta fase solo se tiene que seleccionar dos opciones: *Delete workspace before build starts* y *Add timestamps to the console output*.

A continuación, sigue la configuración de la fase de compilación que se aprecia en la ilustración 40, aquí se puede añadir *Build Steps*, que son bloques en los cuales se ejecuta una serie de código comandos que serán previamente especificados. Existen una gran variedad de tipos de *Build Steps* y se pueden añadir más utilizando *plugin*. Dentro del ámbito de este proyecto se usará la *Execute Shell* donde se define una serie de comandos a ejecutar. Por cómo se encuentra organizado el repositorio, habrá que moverse a la carpeta donde se aloja el código fuente dentro del repositorio. Después, se lanzarán los comandos para poder lanzar las pruebas en *Xcode*, se le indica el dispositivo a probar, además de la versión de *iOS* y se ejecutan las pruebas.

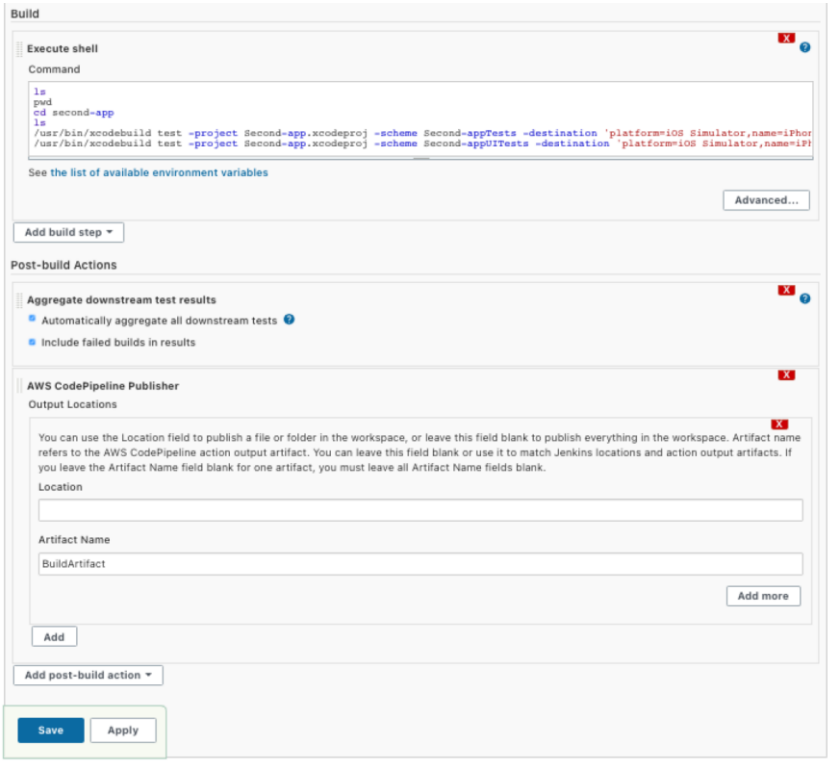


Ilustración 40: Configuración de compilación de un objeto en Jenkins

Finalmente, solo queda la fase de configuración *Post-Build Actions,* donde se pueden configurar los eventos que ocurrirán después de la compilación. Al igual que ocurrió en la fase anteior, existen una gran variedad de eventos que se pueden usar y las posibilidades se aumentan añadiendo *plugin*. Para cumplir con el propósito de este proyecto será necesario añadir las siguientes opciones: *Aggreate Downstream test result* y *AWS CodePipeline Publisher*. Dentro de esta última se le asigna un nombre al artefacto de salida de la acción.

Con esta configuración ya quedaría finalizada esta fase. Para probarla, solo haría falta hacer un *commit* en la rama que enlazamos la canalización en el punto anterior del desarrollo.

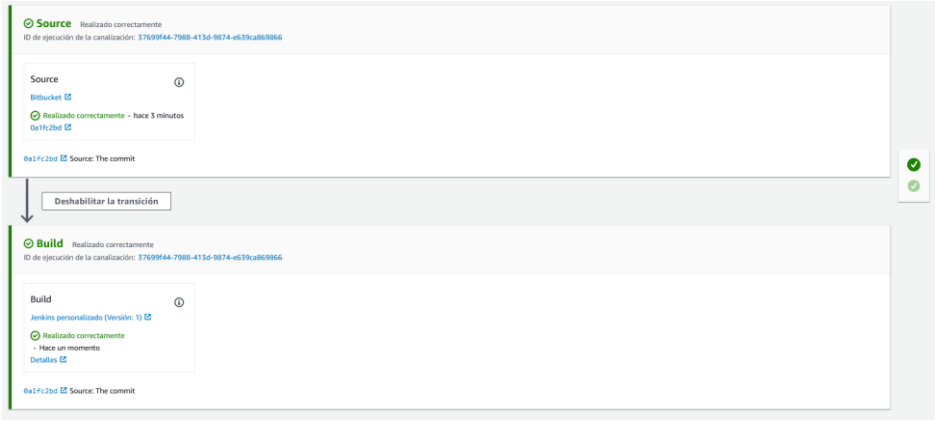


Ilustración 41: Canalización ejecutada

Como se puede ver en la ilustración 41, inicia la ejecución, recoge el artefacto de entrada y ejecuta los comandos de pruebas a la perfección. Desde *Jenkins* se encuentra el registro de la consola y también se encuentran las pruebas que se han ejecutado. También se guarda el artefacto de salida en el *bucket* de *S3*.

7.2.3. Despliegue de *ReportPortal*

Durante esta parte del desarrollo se desplegará una instancia de *ReportPortal*, para ello se utiliza una instancia *EC2* de *Amazon Web Services*. Es necesario recalcar que, de todas las formas en las que se puede desplegar *ReportPortal,* se hará mediante *Docker*, usando como sistema operativo *Ubuntu*.

Se empieza accediendo al servicio de *EC2* en la consola de *AWS*, se accederá a las instancias y se selecciona lanzar una. Se le asigna un nombre, dentro de las imágenes se escoge Ubuntu y se selecciona la arquitectura de 64 bits.

Lo siguiente sería escoger el tipo de instancia que le asignaremos. Para ello se hablará brevemente sobre los requisitos y rendimiento de esta herramienta. En la documentación se puede ver como recomiendan una serie de configuraciones dependiendo de la escala del sistema: empezando con un mínimo de 4 núcleos de procesamiento, 8GB de memoria y 300GB de almacenamiento hasta un máximo de 32 núcleos de procesamiento, 64GB de memoria y 2000GB de almacenamiento.

Los recursos son necesarios para ejecutar una serie de servicios internos dentro de *ReportPortal*. Toda la interfaz gráfica está diseñada y programada en *React UI*. Usa *RabbitMQ,* que es sistema de comunicación mediante paso de mensajes que le permite funcionar a través de eventos. *Elastic Search* como motor de búsquedas. *PostegreSQL* como base de datos. Por último, *Minio* para el sistema de archivos.

Se ha escogido una configuración intermedia para este proyecto, con 4 núcleos de procesamiento, 16Gb de memoria y 500Gb de almacenamiento. La ventaja que brinda *EC2* es que se puede monitorizar la instancia y aumentarle los recursos bajo demanda, para que bajo una carga muy pesada, el servicio se pueda ejecutar sin problemas. Esto proporciona una gran flexibilidad y permite reducir los costes.

Después, para seguir con la configuración de la instancia, se crean un par de claves para el inicio de sesión y se seguirán los mismos pasos que se usaron cuando lanzamos la instancia de *MacOS*.

A continuación, en la configuración de red se dejarán las opciones: *VPC*, Subred y asignar *IP* automáticamente por defecto. Dentro del cortafuegos se le define un nombre, descripción y se describirán las normas que definirán las conexiones que pueden acceder a la instancia. Se le aplicará una norma para permitir las conexiones *SSH* y otra para poder usar *ReportPortal* a través del puerto 8080.

Por último, para acabar la configuración de la instancia, se asignará un almacenamiento. Como se explicó anteriormente se le asignarán 500GB para que pueda ejecutarse sin problemas.

Una vez acabada la configuración, se lanza la instancia. Después de un tiempo de inicio del servicio, será posible establecer una conexión a través del panel de control. Se usará una conexión *SSH* para acceder a la instancia, por lo que se necesitará la clave que generada anteriormente para el inicio de sesión.

Una vez conectados a la máquina, el siguiente paso del proyecto sería instalar *Docker*. Para ello, primero será necesario añadir el repositorio con los siguientes comandos:

```
sudo apt-get update

sudo apt-get install \

    ca-certificates \

    curl \

    gnupg \

    lsb-release


curl -fsSL https://download.Docker.com/Linux/ubuntu/gpg | sudo gpg -
-dearmor -o /usr/share/keyrings/Docker-archive-keyring.gpg


echo \

  "deb        [arch=$(dpkg        --print-architecture)        signed-
by=/usr/share/keyrings/Docker-archive-keyring.gpg]
https://download.Docker.com/Linux/ubuntu \

  $(lsb_release       -cs)       stable"       |       sudo       tee
/etc/apt/sources.list.d/Docker.list > /dev/null
```

Tabla de comandos 1: Añadir Docker

Luego, simplemente se ejecuta el comando de instalación para *Docker Engine* y también se instalará el *Docker Compose*, herramienta que también se usará.

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-
compose-plugin
```

Tabla de comandos 2: Instalar Docker

Lo siguiente sería el despliegue de *ReportPortal*, descargando el archivo *Docker compose* donde está la configuración para lanzar la aplicación.

```
curl -LO
https://raw.Githubusercontent.com/ReportPortal/ReportPortal/master/D
ocker-compose.yml
```

Tabla de comandos 3: Descargar la imagen de ReportPortal

También se tendrá que desplegar *Elastic Search*. Para ello, se necesitan ejecutar los siguientes comandos: el primero es para descargarnos el servicio y el segundo para lanzarlo.

```
Docker pull Docker.elastic.co/elasticsearch/elasticsearch:7.10.2

Docker run -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node"
Docker.elastic.co/elasticsearch/elasticsearch:7.10.2
```

Tabla de comandos 4: Despliegue de ElasticSearch

Luego se crea una carpeta para el servicio *Elastic Search* y se le asignan los permisos necesarios para que el servicio pueda utilizarla con los siguientes comandos:

```
mkdir -p data/elasticsearch
chmod 777 data/elasticsearch
chgrp 1000 data/elasticsearch
```

Tabla de comandos 5: Permisos a las carpetas que usa ElasticSearch

Finalmente, ya se cumplen todos los requisitos para el despliegue, así que ya se puede lanzar el servicio, esto se consigue simplemente ejecutando el siguiente comando:

```
Docker-compose -p ReportPortal up -d --force-recreate
```

Tabla de comandos 6: Lanzar ReportPortal

El servicio tardará un poco en arrancar completamente y una vez pasado un tiempo se podrá acceder mediante el navegador usando la dirección de la máquina y el puerto 8080 donde se aloja el servicio web y mostrará una vista como la de la ilustración 42. De esta forma se puede comprobar que ya se encuentra en ejecución.

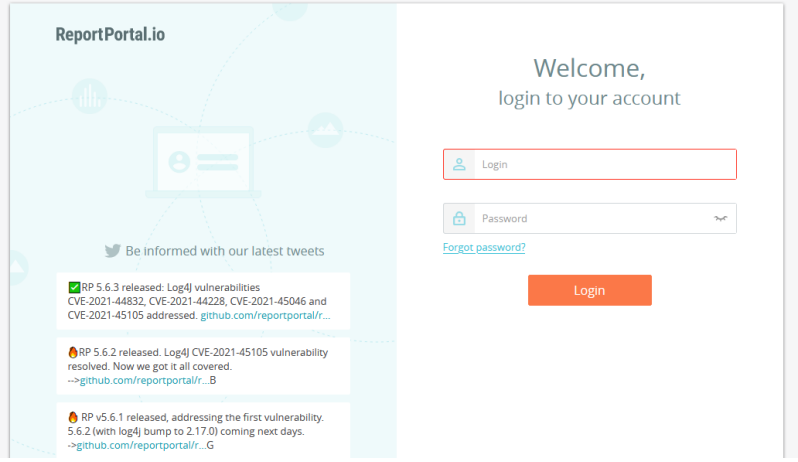


Ilustración 42: Inicio de sesion ReportPortal

Por último, hay que mencionar que el inicio de sesión por defecto se encuentra en la documentación de *ReportPortal*. Una vez dentro se puede añadir un correo a la cuenta, una foto y otros datos. Este usuario tiene todos los permisos. También puede empezar a enviar correos de invitación para que distintos usuarios puedan usar la aplicación, asignarles una serie de roles y administrar sus permisos dentro de la plataforma.

7.2.4. Mejorar implementación del *framework* con *ReportPortal*

Cuando se empezó este proyecto, uno de los objetivos era enriquecer la información que recoge *ReportPortal* para mejorar la experiencia final. Para ello, se pensó en añadir funcionalidades que faltaban dentro del agente, pero que sí estaban latentes dentro de *ReportPortal,* como es la capacidad de añadir etiquetas y descripciones a las distintas pruebas.

Antes de empezar con la mejora, hay que conseguir que el agente funcione. Como punto de partida se probará a seguir las instrucciones del repositorio. Se empezará abriendo un terminal, se selecciona el escritorio donde se encuentra el código fuente de nuestra aplicación y ahí se ejecuta el comando "`pod init`", que creará un archivo *podfile* donde se guardan los distintos *pods* que se instalan y se usan en el proyecto. Se abre este archivo y se modifica añadiendo "`pod 'ReportPortalAgent'`" dentro del *target* de la aplicación. Una vez hecho esto, en la misma consola, se ejecuta el comando "`pod install`" y se instalarán los *pods*.

Una vez acabada la instalación, se abre el archivo con la extensión *xcworkspace* para abrir el proyecto. Se puede ver cómo en los *pods* está instalado el agente de *ReportPortal*. Además, permite ejecutar el proyecto, pero cuando se lanzan las pruebas da un error que se aprecia en la ilustración 43.

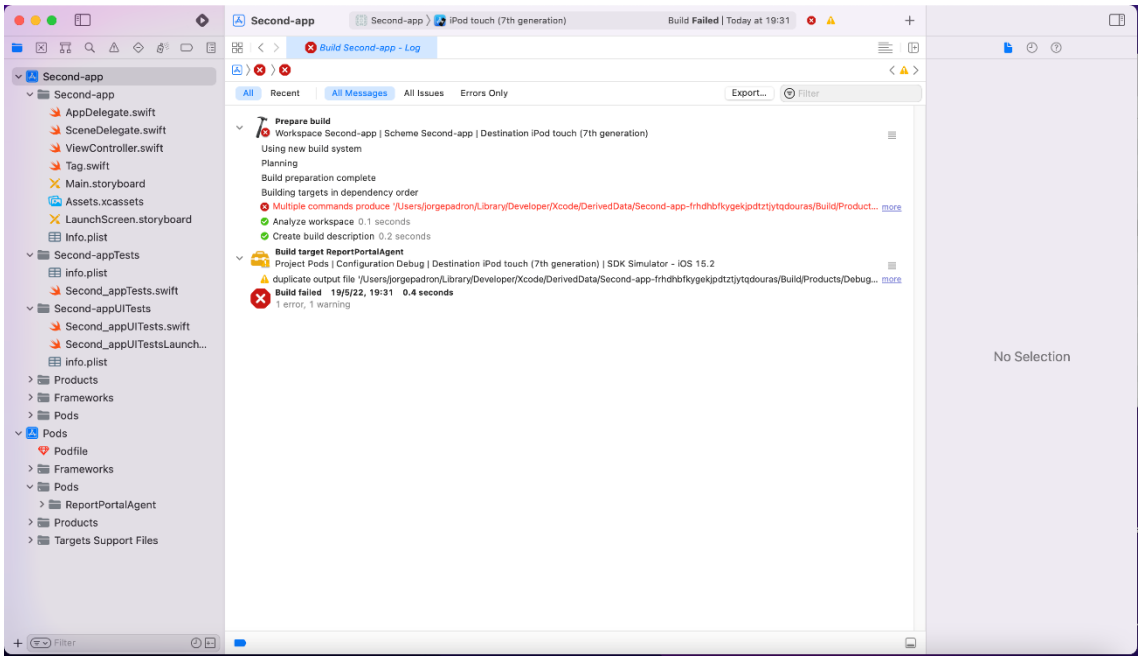


Ilustración 43: Xcode error con pods

Si se busca dentro del repositorio no hay ningún tipo de información sobre el error, tampoco hay ningún *issue* creado con el problema. Además, la descripción del error no brinda mucha información.

Podría ser que la implementación dentro de *CocoaPods* esté desactualizada, así que lo siguiente sería instalar el agente desde una perspectiva diferente. Esta vez se intentará como si fuera un *framework*. Para ello se descarga el código de la librería y se desinstalan los *pods* con el comando "`pod deintegrate`". A partir de aquí se volverá a abrir la aplicación mediante el archivo con la extensión *xcodeproj* por defecto.

Considerado lo anterior, se vuelve a abrir la aplicación para crear un nuevo *framework*. Se hace clic donde pone el nombre de este proyecto para abrir la configuración general, aquí hay que fijarse en la parte izquierda donde se encuentran los *targets*. Se hace clic en el botón del símbolo más para crear un *target* nuevo. A continuación, se busca en la vista generada un *framework*, se selecciona y se le da a siguiente. Ahora pide rellenar una serie campos. Se le da un nombre, y se desactivan las opciones de incluir pruebas y documentación

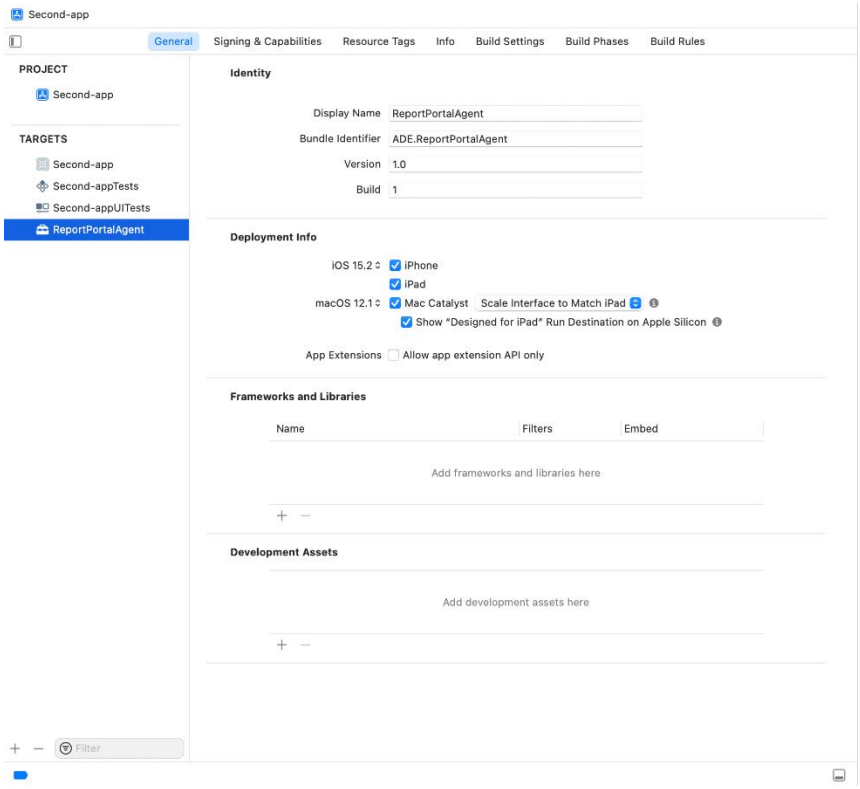


Ilustración 44: Creación del framework

Como se puede apreciar en la ilustración 44, ya se ha generado el *framework*. Este se aloja en una carpeta dentro del proyecto donde ya se puede ver que se encuentra un archivo de cabecera, algo muy habitual en lenguajes como *C/C++*. Ahora, con el código del repositorio del agente ya descargado, se importa manualmente para poder probar si funciona. Del repositorio solo haría falta importar la carpeta *sources* que es donde se aloja el código fuente. Después de hacer esto se intenta compilar el *framework* pero vuelve a generar error, esta vez bastante más descriptivo. Avisa de un posible conflicto entre archivos de configuración al importar el *framework*. Al acceder a la carpeta *sources* del *framework*, se busca el archivo con extensión *plist* y se borra.

Como resultado ya se tiene la capacidad de poder compilar el *framework*, pero ahora se probará a compilar el proyecto entero. Todavía no se ha configurado la conexión del agente con *ReportPortal,* así que simplemente le añadimos al código unas líneas que avise por la consola en que paso se encuentran de la ejecución. Una vez compilado el proyecto sin ningún error, se prueba a ejecutar las pruebas.

A pesar de que se añadieron esas líneas de código, no se ve ninguno de los mensajes que deberían aparecer en la consola. Aparentemente no se está ejecutando el código compilado del agente. Por ello se comprueba la configuración del proyecto para ver si falta ajustar algún

parámetro. Prestando atención a los *targets* de las pruebas, se puede apreciar el problema: no se añade el agente cuando se compilan las pruebas algo que se puede comprobar en la ilustración 45.

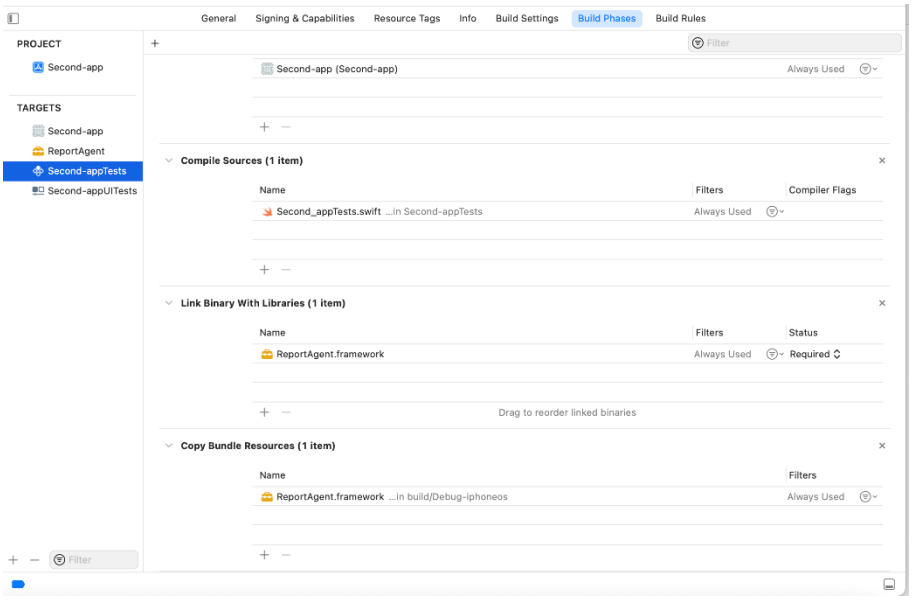


Ilustración 45: Configuración de las fases de compilación en Xcode

Simplemente se buscará dentro de los *targets* de pruebas las secciones *Link Binary With Library* y *Copy Bundle Resources*. Se pulsará el botón de añadir, se busca en la lista *ReportAgent* y se añade. A partir de ahora, cuando se compile la aplicación para las pruebas, se deberá añadir el agente a la compilación para poder ejecutarlo.

```
testBundleWillStart - REPORTAGENT
Test Suite 'All tests' started at 2022-05-20 12:04:35.783
testSuiteWillStart - REPORTAGENT
Test Suite 'Second-appTests.xctest' started at 2022-05-20 12:04:35.788
testSuiteWillStart - REPORTAGENT
Test Suite 'Second_appTests' started at 2022-05-20 12:04:35.789
testSuiteWillStart - REPORTAGENT
Test Case '-[Second_appTests.Second_appTests testExample]' started.
testCaseWillStart - REPORTAGENT
Test Case '-[Second_appTests.Second_appTests testExample]' passed (0.014 seconds).
testCaseDidFinish - REPORTAGENT
Test Case '-[Second_appTests.Second_appTests testPerformanceExample]' started.
testCaseWillStart - REPORTAGENT
/Users/jorgepadron/Desktop/TFG/Proyecto Framework implemnteacion git/second-app/Second-appTest
```

Ilustración 46: Ejecución con el agente

Realizados estos ajustes, comprobamos en la ilustración 46 que el agente funciona, aunque no sea capaz de conectarse con *ReportPortal*. Ahora se abordará otro de los objetivos del proyecto, el enriquecimiento de la información que envía.

Primero, se estudiará cómo están implementadas las etiquetas. Para ello se entra en el repositorio y se comienza a investigar dentro de los distintos archivos. En el agente se puede

ver que los principales archivos donde se ejecuta la mayoría de la lógica son: *RPListener* y *ReportingService*.

Si se buscan referencias a las etiquetas en el primero de ellos se puede encontrar de qué manera carga las etiquetas, que se encuentran guardadas en el archivo de configuración. Simplemente intenta cargarlas y dividirlas por las comas en una matriz compuesta de cadena de caracteres. Estas etiquetas serán asignadas al lanzamiento dentro de *ReportPortal*. Lo que se busca es extrapolar este funcionamiento a las pruebas.

Igualmente se puede ir al *end point* donde se define el contenido de la petición cuando se empieza la ejecución, para poder ver cómo le envían las etiquetas que se encuentran en el archivo de configuración, además de otras etiquetas que consigue desde el método *TagHelper*. Este aporta datos como el nombre del dispositivo que se ejecuta, la versión del sistema, el nombre del modelo y el modelo.

No hay mucho más de la implementación de las etiquetas. En el *end point* de comenzar las pruebas se puede ver cómo directamente las deja vacías. Por otro lado, la implementación de las descripciones es aún más pobre, simplemente dejándolas en vacío en las pruebas y en las ejecuciones.

Se empezará planteando como asignar las etiquetas a las pruebas. Para ello, se define una variable llamada *tags* donde se guarda un diccionario (un conjunto de datos guardados en pares clave/valor). Aquí se guardará en un lado el nombre de la prueba y en el otro las etiquetas que queramos asignar.

Esta variable se le pasa al *end point* junto al resto de datos cada vez que hagamos una llamada para poder enviarla. Esto significa que se tiene que añadir la variable al constructor del *end point* que se envía cuando empieza una prueba. Cuando eso suceda, al constructor habrá que pasarle, como parámetro, las etiquetas. Solamente se deberán enviar las que corresponden a esa prueba y pasarlos como una matriz de cadenas, no como una sola cadena.

```swift
func startTest(_ test: XCTestCase) throws {
  guard let launchID = launchID else {
    throw ReportingServiceError.launchIdNotFound
  }
  guard let testSuiteID = testSuiteID else {
    throw ReportingServiceError.testSuiteIdNotFound
  }

  let endPoint = StartItemEndPoint(
    itemName: extractTestName(from: test),
    parentID: testSuiteID,
    launchID: launchID,
    type: .step,
    arrTags: (testTags[extractTestName(from: test)]?.trimmingCharacters(in:
        CharacterSet.whitespacesAndNewlines).components(separatedBy: ",")) ?? ["Test"]
  )

  try httpClient.callEndPoint(endPoint) { (result: Item) in
    self.testID = result.id
    self.semaphore.signal()
  }
  _ = semaphore.wait(timeout: .now() + timeOutForRequestExpectation)
}

```

Ilustración 47: Implementación de las etiquetas

Como se puede ver en la ilustración 47, lo que se hace es buscar la clave, la cual sería el nombre de la prueba, y se recupera el valor, se eliminan los espacios en blanco y se separa por las comas. Esto se envía con el resto de la petición.

Se planeará la implementación de las descripciones de la misma manera, con una variable que sea un diccionario con los nombres de las pruebas como clave y siendo la descripción el valor. Después se tendrá que editar el paquete que se envía al acabar las pruebas para añadirle la descripción al constructor. La mayor diferencia se encuentra en que, a la hora de recuperar la descripción del diccionario, no necesita ningún tipo de tratamiento, solamente la clave.

Con el código que se ha añadido, el agente es capaz de añadir descripciones y etiquetas a las pruebas dentro del *framework*. Lo siguiente que se hará será interconectar el agente con el resto del proyecto para poder asignarles los valores a los respectivos diccionarios. Se declaran dos variables, una para las descripciones y otra para las etiquetas fuera de la clase en el *ReportingService*. También se crea una función para asignarle valores a estas variables como se puede ver en la siguiente ilustración.

```swift
class Second_appUITests: XCTestCase {

    override class func setUp() {
        super.setUp();
        // Called once before all tests are run
        let desc: [String : String] = [
                    "testDissabledButtons" : "Check if both buttons are disabled",
                    "testInputTextAndTapButton": "Check if i input some text the buttons are touchable",
                    "testGoBackButton": "Test if the go back button function as intended",
                    "testLaunchPerformance":"Test for mesaure the performance"];
        let tags: [String : String] = ["testDissabledButtons" : "UI, Test",
                                       "testInputTextAndTapButton": "UI, Test",
                                       "testGoBackButton":"UI, Test",
                                       "testLaunchPerformance":"UI, Test"]
        ReportAgent.initDescReportingService(descArr: desc, tagsArr: tags)
    }
```

Ilustración 48: Ejemplo de uso de método para inicializar

Como resultado ya se tiene la implementación del agente con las etiquetas y las descripciones. Lo único que se tiene que hacer para usarlo es importar el agente, llamar al método y pasarle los diccionarios como parámetros en la clase donde definamos las pruebas.

7.2.5. Conexión de las pruebas con *ReportPortal*

En este apartado del desarrollo se configurará el agente que se ha modificado, para que sea capaz de conectarse con la instancia de *ReportPortal* que se ha desplegado.

En primer lugar, se vuelve al repositorio donde se encontraba el agente. Allí se encuentra en la documentación los parámetros que se necesitan para que el agente funcione y sea capaz de conectarse.

Se necesita generar un archivo *info.plist* por cada *target*. Será donde se guarden los parámetros de configuración siguientes:

- *ReportPortal*URL: La dirección donde esta alojada la *API* de la instancia de *ReportPortal*.
- *ReportPortal*Token: El *token* que se usará para autenticarnos en la *API*.
- *ReportPortal*LaunchName: Nombre de la ejecución dentro de *ReportPortal*.
- *Principal Class*: La clase donde está definido e implementado el protocolo *XCTestObservation*.
- PushTestDataTo*ReportPortal*: Para activar o desactivar el envió de información a *ReportPortal*.
- *ReportPortal*ProjectName: Nombre de proyecto de *ReportPortal*.
- *ReportPortal*Tags: Este campo es opcional y lo conforman las etiquetas separadas por una coma.
- *IsFinalTest*: Se usa para poder marcar el último test y así que cada *target* tenga su propia ejecución.

Ahora se tiene que generar el archivo de configuración *info.plist* con los campos. Para ello se debe hacer clic derecho en el directorio donde están las pruebas y, seleccionando nuevo archivo, aparecerá una pantalla que permitirá partir de un archivo vacío. Se le da el nombre *info.plist*, se crea y se confirma la extensión del archivo.

Dará un error ya que el archivo no tiene los datos para ser reconocido como un fichero de configuración. Este problema se soluciona dándole un formato al archivo, se abre el archivo como código fuente y se encuentra un archivo XML, se editará el contenido con el siguiente:

Una vez realizados estos ajustes, el sistema lo cataloga como un archivo de configuración, y

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
</dict>
</plist>
```

Tabla 4: Formato XML de un Info.plist

aquí ya se pueden añadir las claves con su respectivo valor. Se destaca que este proceso hay que hacerlo por cada ejecución de pruebas.

Una vez asignadas las claves se necesita el valor de estas. Se consiguen dentro de *ReportPortal*. Lo primero que se tiene que hacer es acceder a la instancia usando su dirección e iniciar sesión dentro de *ReportPortal*.

En la barra lateral derecha se encuentra nuestro usuario, se le hará clic y se selecciona perfil. Como comprobamos con la ilustración 49, aquí se entra en el perfil donde se puede cambiar el nombre de usuario, correo, foto de perfil, idioma y contraseña. Además de eso, también se puede ver el *Access token* y varios ejemplos de configuraciones de agentes en distintos lenguajes como pueden ser: *Java, Ruby, SoapUI, .net* y *NodeJS*.

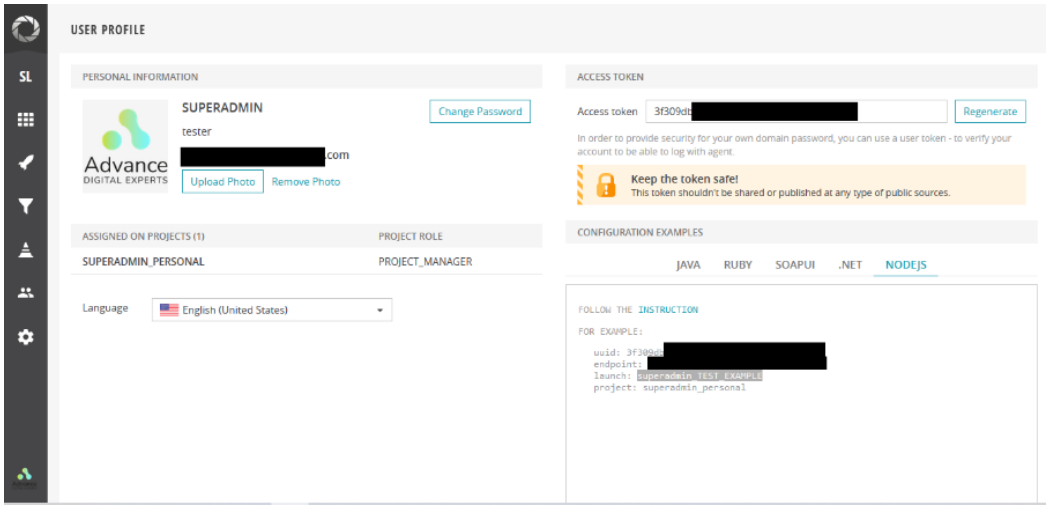


Ilustración 49: Vista de perfil ReportPortal

Se usará el ejemplo de configuración de *NodeJS* para extrapolar la configuración en nuestro archivo *info.plist*. Lo único que se debe hacer es copiar el *UUID*, la dirección del *endpoint* y el nombre del proyecto como vemos en la siguiente ilustración.

Key	Type	Value
Root	Dictionary	(9 items)
PrincipalClass	String	ReportAgent.RPListener
NSPrincipalClass	String	ReportAgent.RPListener
PushTestDataToReportPortal	Boolean	1
ReportPortalLaunchName	String	iOS_Unitary
ReportPortalTags	String	Unitary, Test, SecondApp
ReportPortalToken	String	3f309db1[redacted]
ReportPortalURL	String	[redacted]
ReportPortalProjectName	String	superadmin_personal
IsFinalTestBundle	Boolean	1

Ilustración 50: Configuracion de info.plist

Como resultado se debe tener ya interconectadas las pruebas con *ReportPortal*, así que se procede a ejecutarlo para ver si todo funciona correctamente.

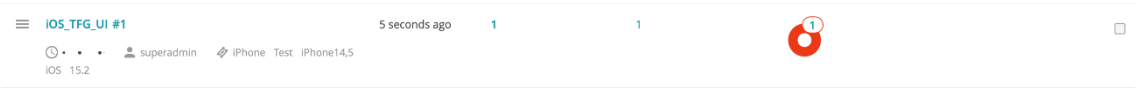
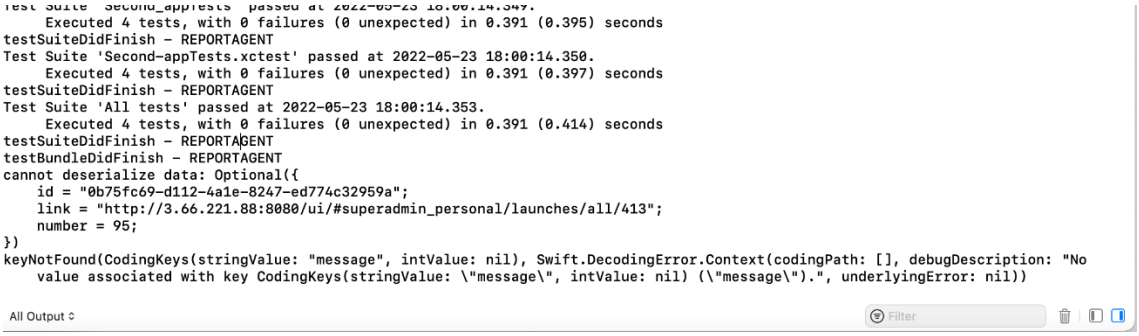


Ilustración 51: Ejecuccion dentro del ReportPortal

Se puede ver cómo aparece en la ejecución en *ReportPortal* pero si se observa la consola de la ejecución en *Xcode* en la ilustración 52 se puede ver que algo falló.



```
     Executed 4 tests, with 0 failures (0 unexpected) in 0.391 (0.395) seconds
testSuiteDidFinish - REPORTAGENT
Test Suite 'Second-appTests.xctest' passed at 2022-05-23 18:00:14.350.
     Executed 4 tests, with 0 failures (0 unexpected) in 0.391 (0.397) seconds
testSuiteDidFinish - REPORTAGENT
Test Suite 'All tests' passed at 2022-05-23 18:00:14.353.
     Executed 4 tests, with 0 failures (0 unexpected) in 0.391 (0.414) seconds
testSuiteDidFinish - REPORTAGENT
testBundleDidFinish - REPORTAGENT
cannot deserialize data: Optional({
    id = "0b75fc69-d112-4a1e-8247-ed774c32959a";
    link = "http://3.66.221.88:8080/ui/#superadmin_personal/launches/all/413";
    number = 95;
})
keyNotFound(CodingKeys(stringValue: "message", intValue: nil), Swift.DecodingError.Context(codingPath: [], debugDescription: "No
    value associated with key CodingKeys(stringValue: \"message\", intValue: nil) (\"message\").", underlyingError: nil))
```


Ilustración 52: Fallo en la consola

Cuando intenta abrir el contenido del paquete de respuesta al evento de acabar la ejecución, da un error. Este error se debe a un posible cambio de la *API* que no se ha visto arreglado en el agente. Se comprueba al archivo de *ReportingService* y dentro se observará la función *finishLaunch*. Se puede apreciar que cuando llama al *endPoint* espera como resultado un objeto *Finish*, que solamente cuenta con un campo: una cadena de caracteres para el mensaje de respuesta. La respuesta que se recibe viene con un identificador, un enlace y un número. Para

arreglar el error se cambia el objeto a recibir *Finish* por el *LaunchFinish,* que sí que incluye estos campos.

```
testCaseWillStart - REPORTAGENT
Test Case '-[Second_appTests.Second_appTests testTagArrOperation]' passed (0.098 seconds).
testCaseDidFinish - REPORTAGENT
Test Case '-[Second_appTests.Second_appTests testTagCreation]' started.
testCaseWillStart - REPORTAGENT
Test Case '-[Second_appTests.Second_appTests testTagCreation]' passed (0.003 seconds).
testCaseDidFinish - REPORTAGENT
Test Suite 'Second_appTests' passed at 2022-05-23 18:18:36.742.
     Executed 4 tests, with 0 failures (0 unexpected) in 0.409 (0.412) seconds
testSuiteDidFinish - REPORTAGENT
Test Suite 'Second-appTests.xctest' passed at 2022-05-23 18:18:36.748.
     Executed 4 tests, with 0 failures (0 unexpected) in 0.409 (0.418) seconds
testSuiteDidFinish - REPORTAGENT
Test Suite 'All tests' passed at 2022-05-23 18:18:36.749.
     Executed 4 tests, with 0 failures (0 unexpected) in 0.409 (0.422) seconds
testSuiteDidFinish - REPORTAGENT
testBundleDidFinish - REPORTAGENT
```

Ilustración 53: Fallo solucionado

Finalmente, como se aprecia en la ilustración 53, si se lanza la ejecución de las pruebas se puede ver que no da ningún tipo de error y ya está conectado con *ReportPortal.*

7.2.6.Creación del informe

Una vez ya se han registrado las pruebas en *ReportPortal,* toca abarcar la generación de un informe que resuma, acumule y formatee toda esta información. Se crearán una serie de paneles de control para poder visualizar la información de una forma más sencilla y rápida. Para esto se tienen que crear unos filtros donde se escogerán todas las ejecuciones que se quieran analizar.

Primero se accede a *ReportPortal* y desde la barra lateral se abre la pestaña de filtrar, una vez dentro se creará un filtro para solo escoger un grupo determinado de pruebas. En este caso las pruebas que tienen que ver con los lanzamientos que ejecutemos. La norma que se usará para filtrar es que las ejecuciones contengan los caracteres *iOS_TFG* como se ve en la siguiente ilustración .

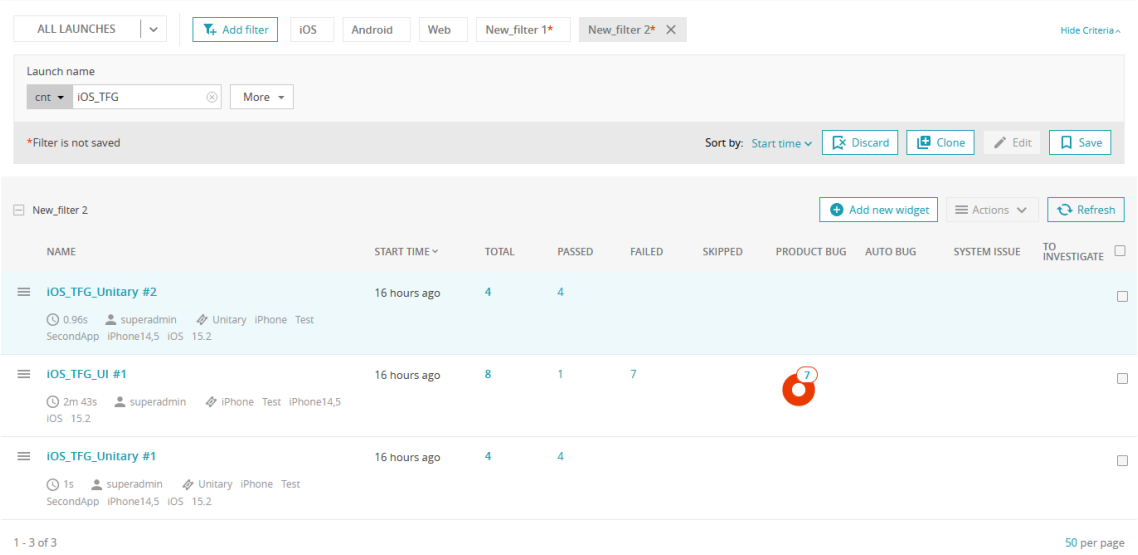


Ilustración 54: Vista de las ejecuciones filtradas

Se guarda el filtro y se le asigna un nombre y descripción. Ahora aparece el filtro creado en la pestaña de filtros y al ser seleccionado muestra una lista de ejecuciones que cumplen el filtro.

El siguiente paso es la creación del panel de control. Se accede a la pestaña desde la barra lateral y se hace clic en crear uno nuevo. Se le asigna un nombre y descripción. Mostrará una pantalla con multitud de opciones.

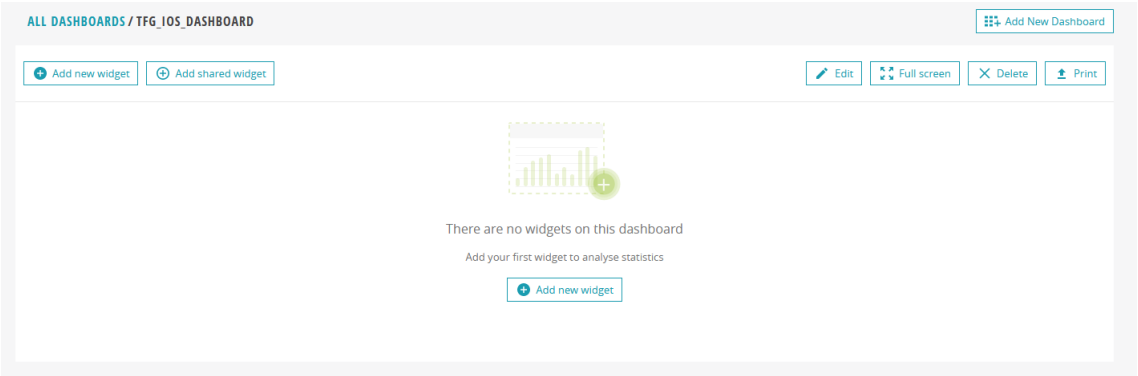


Ilustración 55: Panel de control

Como se puede ver en la ilustración 53, el panel de control se crea vacío. Se tiene que añadir la información que se mostrará mediante el uso de diferentes *widgets*. Así mismo también se puede editar el nombre y la descripción, ponerlo en pantalla completa, borrarlo y por último imprimirlo.

Para añadir un *widget,* se le hace clic en añadir y aparecerá una pantalla donde explica el proceso en tres pasos como se puede ver en la ilustración 56. El primero para seleccionar el tipo, el segundo para configurarlo y un tercero para guardarlo.

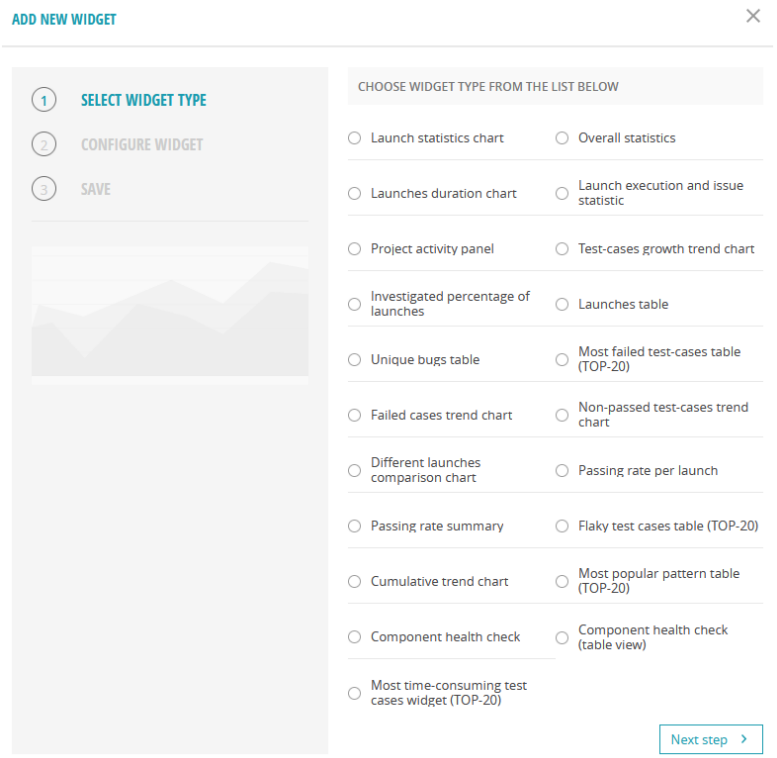


Ilustración 56: Lista de widgets

En el primer paso, se selecciona el tipo de *widget* de una lista con varias opciones donde elegir. Se añadirán los siguientes para la creación del informe: *Launch Statics chart, Overall Statics, Passing rate summary y Launch Tables.*

En el segundo paso de la creación, se eligen los filtros que se usarán para obtener los datos, también se puede definir un criterio para seleccionar las pruebas (por ejemplo, que solo utilice las fallidas), la cantidad de ejecuciones que se usan y las últimas dos opciones permiten editar la apariencia del *widget* y como se mostrará.

El último paso es simplemente una vista previa del *widget*, asignarle un nombre y una descripción y ya se podría añadir. Se seguirá este proceso con los *widgets* anteriormente comentados.

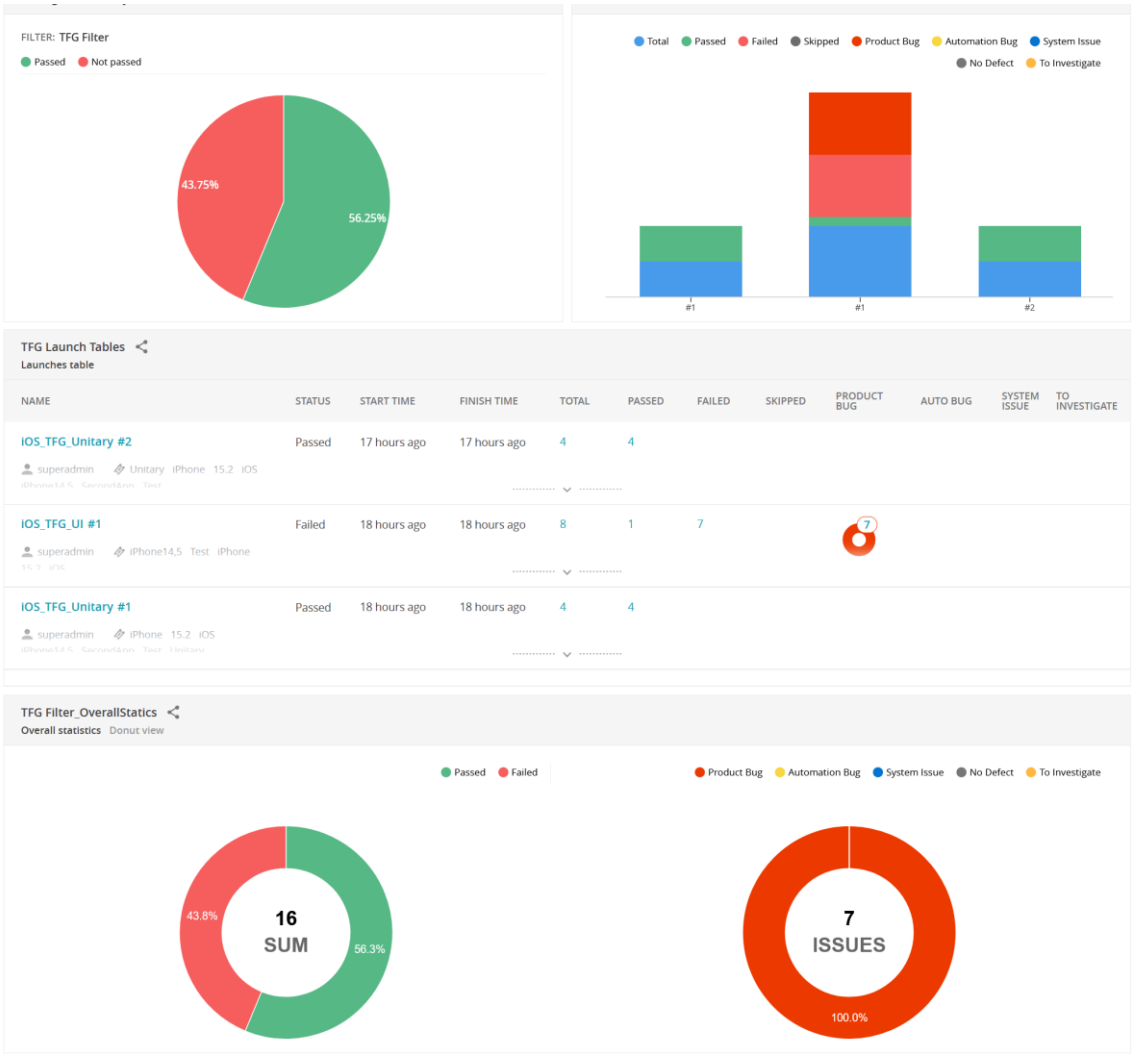


Ilustración 57: Panel de control generado

En la Ilustración 57 se puede ver la vista que se genera pudiendo ver fácilmente la ratio de fallos de las pruebas, las distintas estadísticas de las ejecuciones, una tabla donde se muestran las últimas ejecuciones y debajo unos gráficos con los distintos tipos de fallos de las pruebas. Todos estos elementos son altamente personalizables, se pueden mover, redimensionar, cambiar de nombre, de descripción, la información que muestran y, añadir más elementos.

Finalmente, también se ha de comentar que se puede imprimir. Esto genera un documento con todos los datos mostrados que podría ser guardo en formato PDF para su distribución.

7.3. Validación de las funcionalidades

Para valorar las funcionalidades lo que se hará es simplemente generar un evento de tipo *commit* en una rama del repositorio. Desde ahí se puede observar cómo se ejecutan los distintos servicios que se han configurado.

Empezando por la primera fase de la canalización que empieza cuando llega el *commit,* lo comprime y lo envía al segundo paso de la canalización, que consiste en una fase de ejecución de las pruebas. Aquí habrá que esperar un tiempo a la ejecución de las pruebas dentro de *Jenkins*, en nuestra instancia *EC2* de *MacOS*. Se puede acceder a *ReportPortal* para comprobar que las pruebas se registran.

En este momento se pueden ver los resultados de dichas pruebas, si han fallado o no, cuanto tiempo han tardado y más información útil. Si se accede al panel de control, se puede ver como se ha actualizado la información dentro de los distintos *widgets* incluyendo las nuevas pruebas y resultados.

8. Resultados, conclusiones y trabajos futuros

En este apartado se van a resumir los resultados del desarrollo del servicio, además de remarcar las conclusiones y plantear posibles ideas para mejorar el proyecto de cara al futuro.

8.1. Resultados y conclusiones

Tras la realización de este proyecto se puede decir que el resultado ha sido un éxito. Se ha conseguido desplegar un servicio capaz de integrase a un repositorio, escuchar los eventos de una de las ramas del repositorio, que recupere el código, ejecute pruebas bajo *MacOS* en la nube, que el resultado de las pruebas se recupere y generar un informe con estos resultados. Este servicio puede ser de gran utilidad para muchos desarrollos de aplicaciones y extendiénbdolo podría llegar a ser aplicado para otros entornos de trabajo. También es útil desde el punto de vista de la administración del proyecto, siendo capaz de generar información muy valiosa para conocer el estado del desarrollo, la calidad del producto e incluso el desempeño del equipo de trabajo.

No solamente se han cumplido todos los objetivos del desarrollo, sino personalmente este proyecto le ha servido al autor de este trabajo para desarrollar sus capacidades y conocimientos sobre servicios de *Amazon Web Services*. Sobre todo, herramientas como *Docker* y *Jenkins* que tienen una gran importancia dentro del conjunto de herramientas que debería conocer un desarrollador, además de adentrarse en el campo de la integración y despliegue continuo. Una vez acabado este proyecto sería interesante profundizar más en el ámbito *DevOps* ya que resulta un mundo fascinante con infinidad de posibilidades, de aprendizaje y brinda la capacidad de desarrollarse como futuro profesional.

8.2. Trabajos futuros

En este apartado se comentará cuáles podrían ser las mejoras que se podrían añadir a este proyecto en el futuro con la idea de mejorarlo y hacerlo aún más completo. En el ámbito de este proyecto existen un sin fin de opciones que se podrían desarrollar para la ampliación de las funcionalidades de este servicio. Esto es gracias al uso de *CodePipeline,* que permite personalizar y añadir más etapas dentro de la canalización. Dentro de las múltiples posibilidades que existen, es especialmente interesante comentar la futura incorporación a la canalización de los servicios: *Device Farm*, *Cloud Formatiom* y *SonarQube*.

8.2.1.Device Farm

Device Farm es otro de los múltiples servicios de *Amazon Web Services*. Este se centra en la prueba de aplicaciones web y móviles, ejecutando dichas pruebas en una gran variedad de dispositivos reales tanto *iOS* como *Android* y en multitud de navegadores.

Añadir este servicio a esta canalización aportaría una gran mejora dentro de este proyecto. No es lo mismo ejecutar las pruebas sobre la interfaz gráfica en un dispositivo real que realizarlo en un emulador. El servicio genera también vídeos y registros de cómo se han ejecutado las pruebas dentro el dispositivo. Se destaca que es altamente configurable ya que permite añadir todo tipo de dispositivos y versiones de sistemas operativos.

8.2.2.*Cloud Formation*

Cloud Formation permite predefinir una plantilla con una arquitectura o modelo con los distintos servicios e instancias que se necesitan desplegar, preconfigurados para no tener que crear y configurar manualmente todo cada vez que se necesite lanzar el servicio. Esto permitiría tener la capacidad de lanzar una canalización de forma sencilla.

En el caso que se quiera tener varias canalizaciones para distintas versiones, para distintas ramas del repositorio, para distintas plataformas, etc. También da la habilidad de ser capaz de reducir y aumentar la capacidad del servicio bajo demanda para poder ahorrar costes.

8.2.3.*SonarQube*

SonarQube es una plataforma que permite examinar código fuente para buscar errores, *bugs*, posibles fallos de seguridad y más. Usa servicios de análisis estático de código para evaluar el mismo y es de código abierto.

Usando esta herramienta dentro de la canalización, podríamos ser capaces de ampliar la capacidad de este servicio pudiendo ofrecer un análisis del código que adquirimos del

repositorio, con el objetivo de buscar distintos fallos y así poder mejorar la calidad y seguridad del código. Cuenta con la ventaja que es capaz de analizar código en 29 lenguajes de programación distinto, entre ellos, *Swift*.

9. Bibliografía

Advance Digital Experts. (s.f.). *Homepage*. Recuperado el 24 de Mayo de 2022, de Advance Digital Experts: https://adedelivered.com/

Amazon. (s.f.). *¿Qué es Amazon S3?* Recuperado el 24 de Mayo de 2022, de AWS S3 documentación: https://docs.aws.amazon.com/es_es/AmazonS3/latest/userguide/Welcome.html

Amazon. (s.f.). *¿Qué es AWS CloudFormation?* Recuperado el 24 de Mayo de 2022, de AWS CloudFormation documentación: https://docs.aws.amazon.com/es_es/AWSCloudFormation/latest/UserGuide/Welcome.html

Amazon. (s.f.). *¿Qué es AWS CodeBuild?* Recuperado el 24 de Mayo de 2022, de AWS CodeBuild documentación: https://docs.aws.amazon.com/es_es/codebuild/latest/userguide/welcome.html

Amazon. (s.f.). *¿Qué es AWS CodePipeline?* Recuperado el 24 de Mayo de 2022, de AWS CodePipeline documentación: https://docs.aws.amazon.com/es_es/codepipeline/latest/userguide/welcome.html

Amazon. (s.f.). *¿Que es AWS?* Recuperado el 24 de Mayo de 2022, de AWS: https://aws.amazon.com/es/what-is-aws/?nc1=f_cc

Amazon. (s.f.). *AWS CodeBuild*. Recuperado el 24 de Mayo de 2022, de AWS CodeBuild: https://aws.amazon.com/es/codebuild/?nc=sn&loc=1

Amazon. (s.f.). *Crear una conección con Bitbucket*. Recuperado el 24 de Mayo de 2022, de AWS documentación: https://docs.aws.amazon.com/es_es/dtconsole/latest/userguide/connections-create-bitbucket.html?icmpid=docs_console_unmapped

Amazon. (s.f.). *Integraciones*. Recuperado el 24 de Mayo de 2022, de AWS CodeDeploy: https://aws.amazon.com/es/codedeploy/product-integrations/

Amazon. (s.f.). *Introducción*. Recuperado el 24 de Mayo de 2022, de AWS CodePipeline: https://aws.amazon.com/es/codepipeline/?nc2=h_ql_prod_dt_cp

Amazon. (s.f.). *Introducción*. Recuperado el 24 de Mayo de 2022, de AWS Device Farm: https://aws.amazon.com/es/device-farm/

Amazon. (s.f.). *Introducción*. Recuperado el 24 de Mayo de 2022, de AWS EC2: https://aws.amazon.com/es/ec2/?nc2=h_ql_prod_cp_ec2

Amazon. (s.f.). *Introducción*. Recuperado el 24 de Mayo de 2022, de AWS IAM: https://aws.amazon.com/es/iam/

Amazon. (s.f.). *Introducción*. Recuperado el 24 de Mayo de 2022, de AWS S3: https://aws.amazon.com/es/s3/getting-started/?nc=sn&loc=6&dn=1

Amazon. (s.f.). *Precios de AWS*. Recuperado el 24 de Mayo de 2022, de AWS Pricing: https://aws.amazon.com/es/pricing/

Amazon. (s.f.). *Recursos de AWS para administración de acceso*. Recuperado el 24 de Mayo de 2022, de AWS IAM Documentación: https://docs.aws.amazon.com/es_es/IAM/latest/UserGuide/access.html

Apache. (2004). *Apache License, Version 2.0*. Recuperado el 24 de Mayo de 2022, de Apache: https://www.apache.org/licenses/LICENSE-2.0

Apple. (s.f.). *SwiftUI*. Recuperado el 24 de Mayo de 2022, de XCode: https://developer.apple.com/xcode/swiftui/

Apple. (s.f.). *Testing Your Apps in Xcode*. Recuperado el 24 de Mayo de 2022, de XCTest documentación: https://developer.apple.com/documentation/xcode/testing-your-apps-in-xcode

Apple. (s.f.). *Xcode documentación*. Recuperado el 24 de Mayo de 2022, de Xcode: https://developer.apple.com/documentation/xcode

Apple. (s.f.). *XCTestObservation*. Recuperado el 24 de Mayo de 2022, de XCTest documentación: https://developer.apple.com/documentation/xctest/xctestobservation

Atlassian. (s.f.). *Bitbucket*. Recuperado el 24 de Mayo de 2022, de Atlassian Bitbucket: https://www.atlassian.com/es/software/bitbucket

Brew. (s.f.). *Homepage*. Recuperado el 24 de Mayo de 2022, de Brew: https://brew.sh/

Brookes, P. (16 de Junio de 2020). *Rezaid*. Recuperado el 24 de Mayo de 2022, de The Top 8 Benefits of Choosing Scrum Software: https://rezaid.co.uk/benefits-of-scrum-software/

CocoaPods. (s.f.). *Homepage*. Recuperado el 24 de Mayo de 2022, de CocoaPods: https://cocoapods.org/

Docker. (s.f.). *Install Docker Engine on Ubuntu*. Recuperado el 24 de Mayo de 2022, de Docker documentación: https://docs.docker.com/engine/install/ubuntu/

Git. (s.f.). *Homepage*. Recuperado el 24 de Mayo de 2022, de Git: https://git-scm.com/

https://tightvnc.com/. (s.f.). *TightVNC Homepage*. Recuperado el 24 de Mayo de 2022, de TightVNC : https://tightvnc.com/

Instituto Tecnológico de Massachusetts. (24 de Mayo de 2022). *Wikipedia*. Obtenido de Licencia MIT: https://es.wikipedia.org/wiki/Licencia_MIT

Jenkins. (s.f.). *Homepage*. Recuperado el 24 de Mayo de 2022, de Jenkins: https://www.jenkins.io/

Jenkins. (s.f.). *macOS*. Recuperado el 24 de Mayo de 2022, de Installing Jenkins: https://www.jenkins.io/doc/book/installing/macos/

Kent Beck, M. B. (12 de Febrero de 2001). *Manifiesto ágil*. Recuperado el 24 de Mayo de 2022, de Wikipedia: https://es.wikipedia.org/wiki/Manifiesto_%C3%A1gil

Raj Bala, B. G. (27 de Julio de 2021). *Gartner*. Recuperado el 24 de Mayo de 2022, de Magic Quadrant para servicios de infraestructura y plataforma en la nube: https://www.gartner.com/technology/media-products/reprints/AWS/1-271W1OTA-ESP.html

RedHat. (s.f.). *¿Qué es CI/CD?* Recuperado el 24 de Mayo de 2022, de RedHat.com: https://www.redhat.com/en/topics/devops/what-is-ci-cd

ReportPortal. (30 de Marzo de 2022). *Guias para desarrollador*. Recuperado el 24 de Mayo de 2022, de Github: https://github.com/reportportal/documentation/blob/master/src/md/src/DevGuides/reporting.md

ReportPortal. (s.f.). *Homepage*. Recuperado el 24 de Mayo de 2022, de ReportPortal: https://reportportal.io/docs/

ReportPortal. (s.f.). *Instalación*. Recuperado el 24 de Mayo de 2022, de ReportPortal: https://reportportal.io/installation

ReportPortal. (s.f.). *Repositorio ReportPortal*. Recuperado el 24 de Mayo de 2022, de Github: https://github.com/reportportal/reportportal/tree/5.7.0

Richard Stallman . (24 de Mayo de 2022). *Wikipedia*. Obtenido de Copyleft: https://es.wikipedia.org/wiki/Copyleft

Seisdedos, J. P. (s.f.). *ATP iOS*. Recuperado el 24 de Mayo de 2022, de Bitbucket: https://bitbucket.org/adedelivered/atp-ios/src/development/

Seisdedos, J. P. (s.f.). *ReportAgent_iOS*. Recuperado el 24 de Mayo de 2022, de Bitbucket: https://bitbucket.org/adedelivered/reportagent_ios/src/main/

SonarSource. (s.f.). *Homepage*. Recuperado el 24 de Mayo de 2022, de SonarQube: https://www.sonarqube.org/

Swift. (s.f.). *Welcome to Swift.org*. Recuperado el 24 de Mayo de 2022, de Swift.org: https://www.swift.org/