Universitat de les Illes Balears

Escuela Politécnica Superior

Memoria del Trabajo de Fin de Grado

Plataforma de integración y despliegue continuo

Sergi Muntaner Ruiz

Grado de Ingenieria Informática

Año académico 2018-19

DNI del alumno: 43459705D

Gabriel Moyà Alcover
Departamento de Ciencias de la Computación e Inteligencia Artificial

GRADO DE INGENIERIA INFORMÁTICA

Plataforma de integración y despliegue continuo

SERGI MUNTANER

Tutor
Gabriel Moyà Alcover

Escola Politècnica Superior
Universitat de les Illes Balears
Palma, Marzo de 2019

ÍNDICE GENERAL

ACRÓNIMOS

TFG Trabajo de Fin de Grado

CI/CD integración continua y despliegue contínuo

SCM Source Code Management

C# C Sharp

VB Visual Basic

SO Sistema Operativo

APT Advanced Packaging Tool

RPM Red Hat Package Manager

SCP Secure Copy

IAC Infraestructura como Código

RESUMEN

En este trabajo hemos establecido un método para hacer que las organizaciones que realizan software sean mas competitivas mediante un sistema con el cual pueden integrar sus cambios en el código mas frecuentemente y con mayor calidad.

El sistema que se ha llevado a cabo en este trabajo de fin de grado es una plataforma de integración y entrega continua de software. Esta plataforma consiste en un sistema de automatización que se encarga de realizar automáticamente el ciclo de despliegue sobre cualquier cambio realizado en un programa. Este ciclo consiste en desplegar el código que se integra en una organización llevando a cabo una serie acciones y procesos que aseguran la calidad del código y la seguridad del despliegue. Estas acciones y procesos que forman el ciclo de despliegue están recogidos en el ISO/IEC 12207, el cual expone en el apartado 6.1.2 un conjunto de buenas prácticas referentes al proceso de entrega de software.

Para implementar este sistema se utilizarán un conjunto de componentes, tecnologías y prácticas ligadas a las metodologías ágiles y que se acogen a la metodología DevOps, la cual consiste en la unión de los procesos de desarrollo con los procesos de sistema.

CAPÍTULO 1

INTRODUCCIÓN

El número y el tamaño de las compañías que utilizan las tecnologías de la información (TI) como base de su negocio han ido aumentando drásticamente con el paso del tiempo. Este crecimiento ha causado una gran competitividad en el sector, la cual ha provocado que las empresas tengan la necesidad de adoptar una postura de aceptación al cambio y una gran capacidad de adaptación.

En este trabajo nos vamos a centrar en las empresas que desarrollan software, y vamos a buscar un método para hacer que éstas sean mas competitivas. Para ello tomaremos como objetivo diseñar un sistema para que puedan entregar software mas frecuentemente y con mayor calidad. Para poder ver este proceso y sus resultados tendremos que seleccionar un lenguaje de programación sobre el cual funcionará nuestro sistema e implementar una pequeña aplicación con su respectivo conjunto de pruebas. Sobre esta aplicación se automatizarán los procesos de integración y de entrega continua.

El proceso de integración continua consiste publicar continuadamente en el sistema de almacenamiento de código de la organización los cambios de software realizados, mientras que la entrega continua consiste en el proceso de desplegar automáticamente el código integrado mediante la integración continua en un servidor para que el programa pueda ser consumido. Todo el proceso desde que se integra un cambio hasta que se despliega lo consideramos el ciclo de despliegue, el cual es la finalidad principal de nuestro sistema.

Este sistema se fundamenta en la idea de promover un modelo de implementación ágil en contra del modelo de implementación de software basado en cascada. Gracias al desarrollo iterativo que nos proporciona la metodología Agile podremos realizar entregables mas pequeños que aporten valor en cualquier fase del proyecto. De la otra manera únicamente aportaríamos valor en las fases finales del mismo.

La siguiente imagen representa de forma gráfica las iteraciones de los modelos Ágile en contra la rigidez de los modelos en cascada:

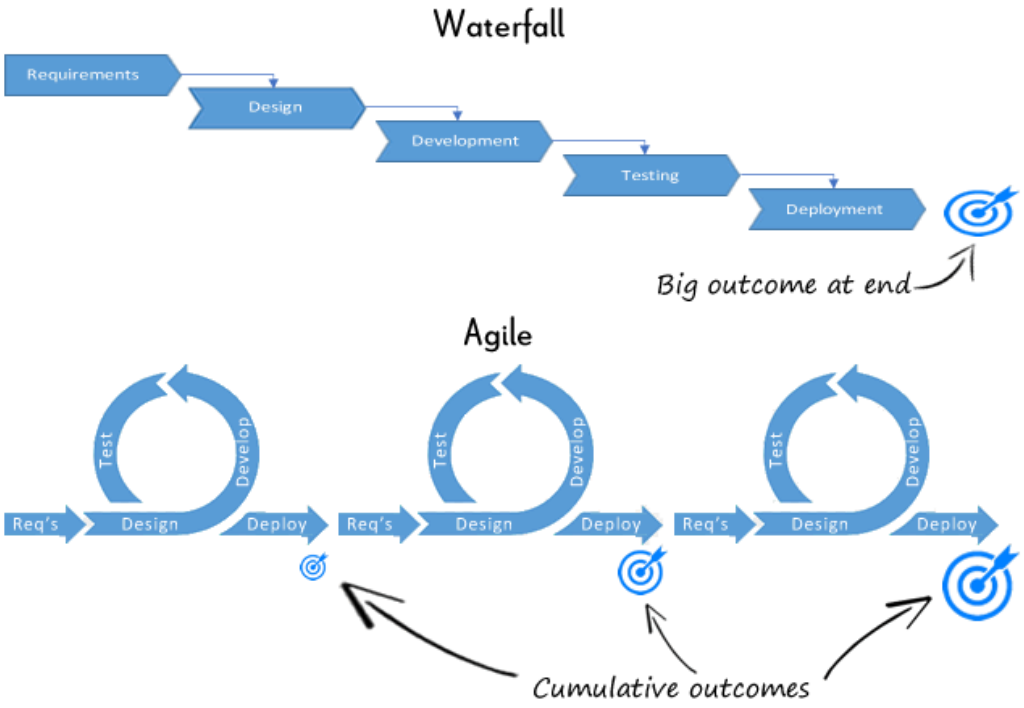

Figura 1.1: Método en cascada vs Agile[1].

Otra base sobre la cual se sustenta este sistema es la metodología DevOps. El término DevOps es "una combinación de filosofías culturales, prácticas y herramientas que incrementan la capacidad de una organización de proporcionar aplicaciones y servicios a gran velocidad: desarrollar y mejorar productos con mayor rapidez que las organizaciones que utilizan procesos tradicionales de desarrollo de software y administración de la infraestructura"[2].

Esta metodología propone un cambio cultural que consiste en romper la separación departamental entre el desarrollo de software y las de operaciones de sistema. Sin esta rotura estaríamos forzados implícitamente a trabajar por fases ya que cuando los programadores acaban la fase de desarrollo se tiene que pasar a la fase de despliegue realizada por administradores de sistemas. De este modo podemos ver que el modelo DevOps promueve las metodologías Agile, ya que la unión del desarrollo de software junto a las operaciones del sistema bajo un mismo departamento o grupo de personas nos permite seguir los siguientes principios del manifiesto Agile[3]:

- Equipos auto-organizados.
- Despliegues frecuentes.
- Conversación cara a cara.
- Entrega continua.

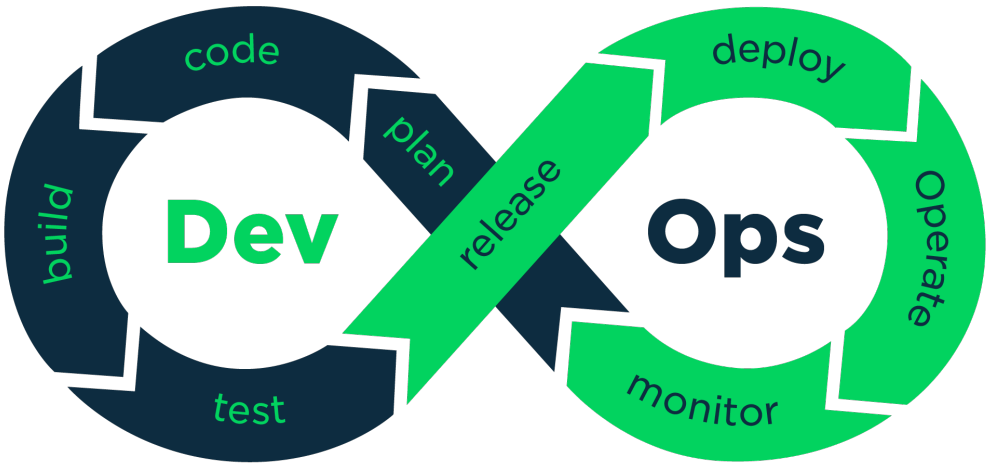

Figura 1.2: Modelo DevOps[4].

El sistema que llevaremos a cabo en este trabajo de fin de grado es una plataforma de integración y entrega continua de software; de aquí en adelante nos referiremos este sistema como plataforma de integración continua y despliegue contínuo (CI/CD). Una plataforma de CI/CD consiste en un sistema de automatización que se encarga de realizar el ciclo de despliegue automáticamente a cualquier cambio realizado sobre un programa de software. Este ciclo consiste en desplegar automáticamente el código integrado en la organización llevando a cabo una serie acciones automáticas para para asegurar la calidad del código y la seguridad del despliegue de un programa. Para implementar este sistema se utilizarán un conjunto de componentes, tecnologías y prácticas ligadas a las metodologías ágiles y que se acogen a la metodología DevOps. También nos apoyaremos en el ISO/IEC 12207[5] el cual recoge un conjunto de buenas prácticas referentes a los procesos del ciclo de vida del software.

Con la realización de una plataforma de CI/CD queremos abordar los siguientes objetivos:

- Reducir el tiempo de lanzamiento al mercado de los proyectos software.
- Dotar a los desarrolladores de un proceso para llevar a cabo sus despliegues.
- Dar a conocer tendencias como las metodologías ágiles y el modelo DevOps.

Este Trabajo de Fin de Grado (TFG) lo vamos a plantear como si se tratase de un proyecto de software, por lo que lo vamos a dividir en los siguientes capítulos:

- **Análisis:** En este capítulo analizaremos como vamos a llevar a cabo nuestro sistema y que acciones debe de llevar a cabo para cumplir nuestros objetivos.
- **Diseño:** En el capítulo de diseño trabajaremos identificando las diferentes tecnologías o componentes que deben formar nuestro sistema. Finalmente seleccionaremos una herramienta específica para cada componente con lo cual formaremos un diseño inicial de nuestro sistema.
- **Implementación:** Una vez establecido el diseño del sistema, procederemos a su materialización en este capítulo. En este capítulo haremos hincapié en las acciones concretas que lleva a cabo cada herramienta y explicaremos a alto nivel la configuración de estas herramientas para poder establecer todo el proceso de despliegue.
- **Resultado:** Cuando tengamos implementado todo el sistema únicamente nos quedará mostrar el resultado de de nuestro esfuerzo y asegurarnos de que éste cumpla nuestros requisitos. Para ello realizaremos una simulación de todo el sistema desde el punto de vista del usuario, y analizaremos que se cumplan todos los requisitos.
- **Conclusión:** Para acabar realizaremos una conclusión final del trabajo aportado, aportando un punto de vista subjetivo al TFG realizado.

CAPÍTULO 2

ANÁLISIS

A continuación vamos a llevar a cabo un análisis para poder definir cuales son los objetivos y criterios de aceptación de nuestra plataforma de CI/CD. Para ello realizaremos la identificación de las partes interesadas del sistema, la extracción de los requisitos y un diagrama de casos de uso.

La realización de este análisis es un paso de vital importancia para el proyecto ya que nos sirve como guía para llevar a cabo nuestro sistema basándonos en unos objetivos. Esto nos ayudará a describir las acciones y interacciones que éste tiene que llevar a cabo.

2.1 Identificación de las partes interesadas

Las partes interesadas de un proyecto son aquellas entidades que se pueden ver afectadas por nuestro sistema. En nuestro caso podemos encontrar tres tipos de usuarios que representarían estas partes interesadas:

- **Desarrollador de software:** éste es el principal interesado de este sistema. Estos usuarios son los que se benefician de este sistema ya que les otorga la capacidad de desplegar los cambios que realizan en el código sin tener que depender de otros usuarios.

- **Súper usuario**: Este usuario es el encargado de crear y configurar todo el sistema. En el contexto de nuestro TFG, este usuario seríamos nosotros ya que somos los que vamos a implementar el sistema

- **Consumidor de la aplicación:** Es el usuario que consume los programas que desplegamos mediante nuestro sistema. Este interesado es el menos relevante ya que el objetivo de este TFG no reside en los programas realizados, sino en los procesos y sistemas utilizados para ponerlos en funcionamiento.

2.2 Requisitos funcionales

Inicialmente se identificarán los requisitos funcionales de nuestra plataforma. Los requisitos funcionales "describen el comportamiento e información que la solución administrará. Describen las capacidades que el sistema deberá poder llevar a cabo en términos de su comportamiento, asociado con acciones de o respuestas específicas a las aplicaciones de tecnologías de la información"[6].

Autenticarse en el sistema

- Código: RF-1.
- Descripción: El usuario tiene que poder autenticarse en los componentes del sistema.
- Proceso: El usuario debe poder autenticarse en los diferentes componentes que forman la plataforma de CI/CD.

Almacenar código

- Código: RF-2.
- Descripción: El sistema debe permitir almacenar el código de los programas.
- Proceso: El sistema tiene que guardar el código fuente de los programas de la organización.

Descargar código

- Código: RF-3.
- Descripción: El usuario debe poder descargarse el código de los programas del sistema.
- Proceso: El usuario autenticado tiene que ser capaz de descargar el código de los programas pertenecientes a la organización sobre los cuales tenga permisos.

Subir código

- Código: RF-4.
- Descripción: El usuario debe poder subir los cambios realizados en el código de los programas al sistema.
- Proceso: El usuario autenticado tiene que ser capaz de subir al sistema sus cambios realizados en el código de los programas sobre los cuales tenga permisos.

Iniciar ciclo de despliegue

- Código: RF-5.
- Descripción: El usuario tiene que poder iniciar el ciclo de despliegue.
- Proceso: El usuario debe poder iniciar el proceso de CI/CD al realizar una subida de código .

Ver el estado del ciclo del ciclo de despliegue

- Código: RF-6.
- Descripción: El usuario ha de ser capaz de ver el estado del ciclo de despliegue.
- Proceso: El usuario autenticado tiene que poder visualizar en el navegador el progreso del ciclo de despliegue. El sistema tiene que mostrar las fases acabadas y su estado de finalización junto a la fase en progreso.

Ejecutar pruebas funcionales de código automáticas

- Código: RF-7.
- Descripción: El sistema debe lanzar las pruebas funcionales incluidas en el código del proyecto.
- Proceso: Cuando el usuario inicia el proceso de CI/CD, se deben lanzar automáticamente la batería de pruebas adjunta en el proyecto. Esta acción es una práctica recogida en el apartado 7.2.3 de la el ISO 12207[5].

Visualizar el resultado de las pruebas funcionales

- Código: RF-8.
- Descripción: El sistema debe permitir la visualización del resultado de las pruebas.
- Proceso: El usuario autenticado debe poder visualizar vía web el resultado de la batería de pruebas.

Generar artefacto de código

- Código: RF-9.
- Descripción: El sistema debe ser capaz de generar artefactos de código.
- Proceso: Al ser satisfactoria la batería de pruebas del código integrado en el repositorio, el sistema debe generar un artefacto de código. En caso de que la batería de pruebas no sea satisfactoria no se deberá generar el artefacto de código. Un artefacto de código es el conjunto de binarios y dependencias que necesita un programa para ejecutarse, por ejemplo si un binario necesita un fichero de texto con cierta información para ejecutarse ambos objetos formarían un artefacto.

Almacenar artefactos de código versionado

- Código: RF-10.
- Descripción: El sistema debe ser capaz de almacenar artefactos de código versionados.
- Proceso: Al generarse un artefacto de código, el sistema lo versionara renombrando el archivo con un contador incremental y lo almacenará en el sistema.

Desplegar artefacto de código automáticamente

- Código: RF-11.
- Descripción: El sistema debe ser capaz de desplegar artefactos de código automáticamente.
- Proceso: El sistema debe desplegar un artefacto de código en un servidor cuando éste es correctamente añadido al repositorio de artefactos.

Dar de alta nuevos usuarios

- Código: RF-12.
- Descripción: El súper usuario tiene que poder dar de alta nuevos usuarios.
- Proceso: El súper usuario debe tener la capacidad de dar de alta nuevos usuarios en los diferentes componentes del sistema.

Acceso a los programas desplegados

- Código: RF-13.
- Descripción: El consumidor de la aplicación tiene que poder acceder a los programas desplegados.
- Proceso: El consumidor de la aplicación debe poder acceder a los programas desplegados por el sistema utilizando su navegador.

2.3 Requisitos no funcionales

Una vez vistos los requisitos funcionales veremos los no funcionales, los cuales “capturan las condiciones que no están directamente relacionadas con el comportamiento o funcionalidad de la solución, por el contrario, describen condiciones del medio en las cuales la solución deberá permanecer eficaz o cualidades que el sistema debe tener"[6].

Cambios nominales en el código

- Código: RNF-1
- Descripción: Los cambios en el código han de estar identificados por el usuario que los realiza.

Cambios identificados

- Código: RNF-2
- Descripción: Los cambios en el código han de estar identificados con un identificativo único.

Tiempo del ciclo de despliegue inferior a cinco segundos

- Código: RNF-3
- Descripción: El tiempo de todo el ciclo de despliegue debe ser inferior a 3 minutos.

2.4 Diagrama de casos de uso

A continuación mostraremos un diagrama con los casos de uso que debería contemplar nuestro sistema. De este modo podremos ver con mas detalle las acciones que puede llevar a cabo cada interesado.

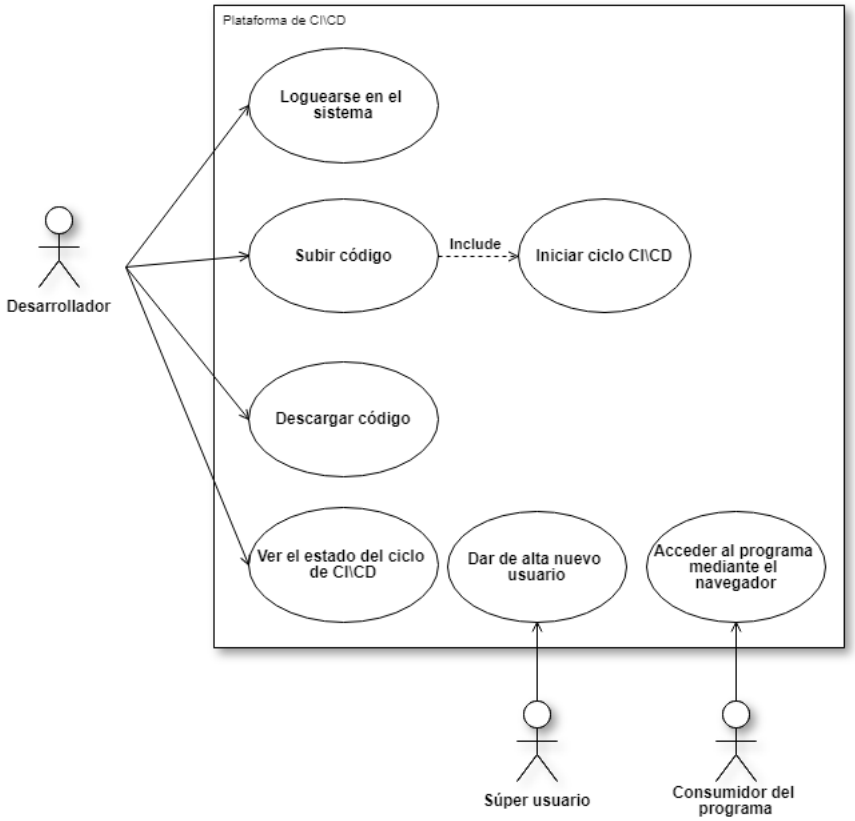

Figura 2.1: Casos de uso de la plataforma de CI/CD.

Con el caso de uso anterior podemos ver cuales son las acciones que el usuario puede llevar a cabo en nuestro sistema pero seguimos sin ver exactamente el proceso que éste lleva a cabo para desplegar el código una vez se inicia el ciclo de despliegue. Por este motivo vamos a complementar el caso de uso anterior con un diagrama de flujo, mediante el cual podremos ver cuales son las acciones que el sistema lleva a cabo automáticamente cuando el usuario inicia el proceso subiendo su código:

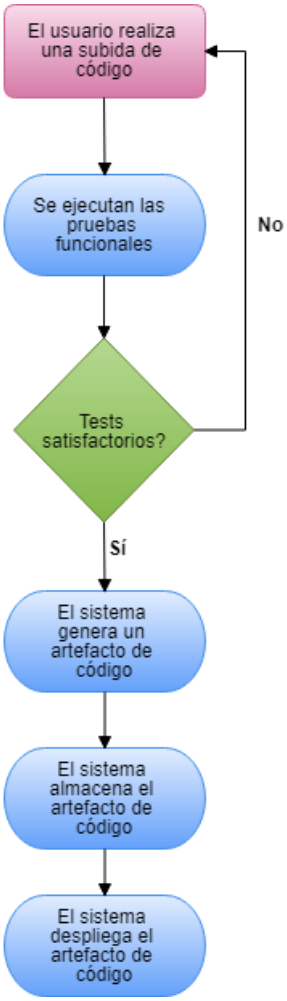

Figura 2.2: Diagrama de flujo del proceso de CI/CD llevado a cabo por el sistema.

Todas las acciones que lleva a cabo el sistema las hemos extraído del apartado 6.1.2 de la ISO 12207[5] que trata sobre el proceso de entrega de software.

2.5 Planificación

Pese a que en este TFG se fundamenta en metodologías ágiles, nosotros vamos a llevar a cabo la planificación del proyecto de un modo más tradicional utilizando un modelo en cascada. Las razones por las que utilizamos un modelo en cascada y no uno Agile son las siguientes:

- Uno de los principales inconvenientes de la metodología en cascada es que es difícil realizar un análisis de requisitos lo suficientemente claro como para dar soporte al resto de fases del desarrollo del proyecto. Muchas veces los requisitos proporcionados por el cliente cambian o simplemente no son lo suficientemente específicos. En este TFG nosotros mismos definimos los requisitos y implementamos el sistema, con lo cual este inconveniente no nos afecta.

- En el contexto de realización de este TFG no hay un cliente al cual involucrar en el desarrollo o aportarle información continuamente, de manera que trabajar de un modo iterativo dando información al cliente no acaba de tener sentido.

- En este proyecto tampoco nos vemos afectados por una de las mayores ventajas de la metodología Agile, que es la de entrega de valor continuo. Si existiera un cliente real podríamos entregar el proyecto en fases pequeñas lo cual le aportaría valor de manera continuada, pero al no existir ningún cliente final que se tenga que beneficiar del sistema no nos es relevante si entregamos el proyecto en fases o completo.

Inicialmente para realizar la planificación hemos desglosado las tareas a realizar y hemos identificaremos los distintos hitos. Al no saber desde un principio que componentes vamos a necesitar para implementar nuestro sistema, las tareas están basadas en las acciones llevadas a cabo por el ciclo de despliegue mostradas en el diagrama de flujo anterior. Todas estas tareas se han documentado en un una herramienta de gestión llamada Trello, y las podemos ver en la siguiente imagen:

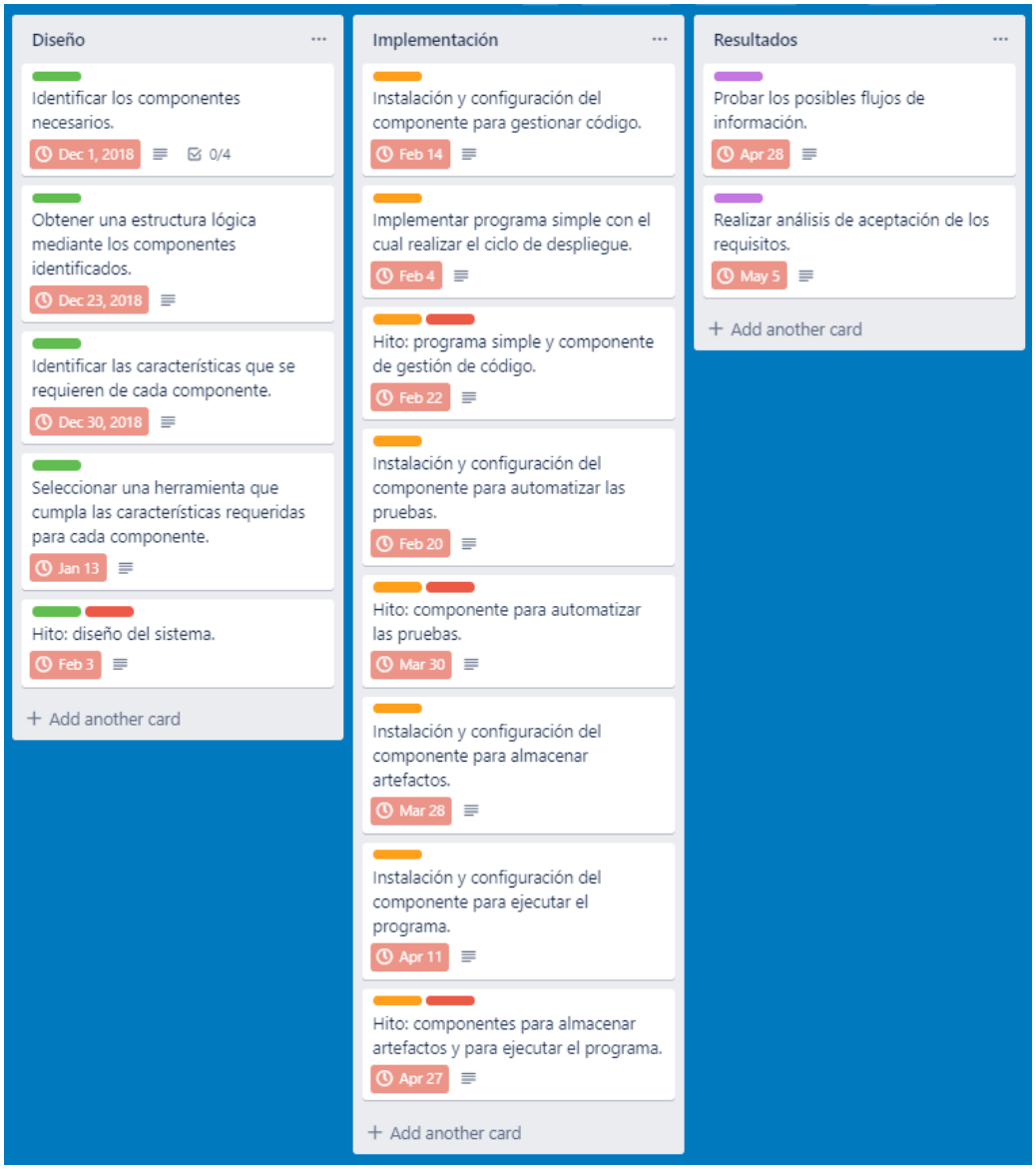

Figura 2.3: Trello con las tareas identificadas para la realización de la plataforma de CI/CD.

A continuación hemos realizado una planificación temporal, en la cual se ha establecido que el proyecto empieza a principios de Diciembre del 2018 con las tareas propias de la fase de diseño, y acaban a mitad de Mayo del 2019 con las tareas de llevar a cabo los resultados.

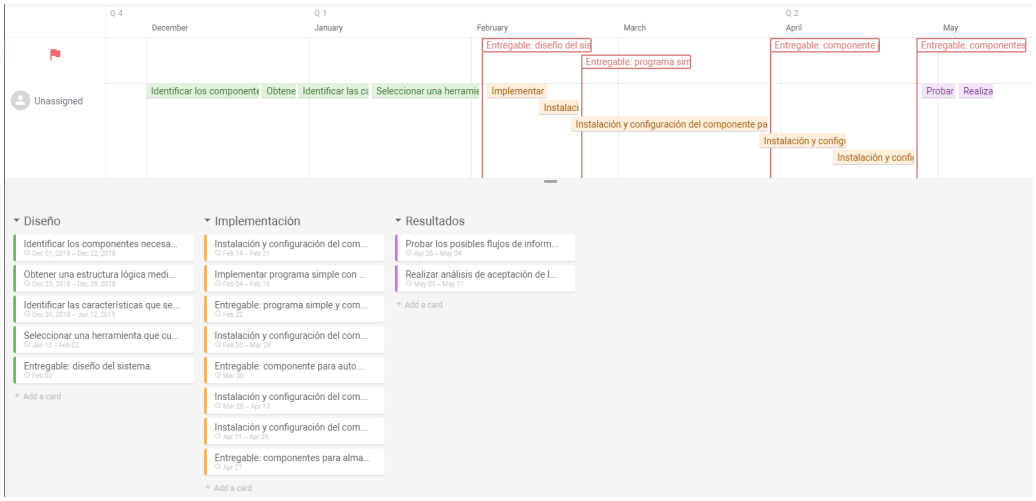

Figura 2.4: Planificación temporal de las tareas.

2.6 Conclusiones del análisis

Podemos observar que con el análisis realizado tenemos bien definidas las acciones que forman el ciclo de despliegue de una aplicación desde que se sube un cambio en un programa hasta que se pone en producción, estableciendo mecanismos que aseguran la calidad y seguridad de nuestro código como lanzar una batería de pruebas contra el programa o almacenar los artefactos generados para cubrir el caso de que si el despliegue de un artefacto no es satisfactorio podamos ir rápidamente a un artefacto anterior.

Una vez llevado a cabo el análisis para definir que tareas tiene que llevar a cabo nuestra plataforma de CI/CD vamos a empezar a establecer el diseño del sistema acorde a ellos. Lo que haremos en el siguiente apartado es detectar que componentes vamos a necesitar para diseñar una estructura que sea acorde a nuestros requisitos y a continuación buscaremos un producto específico para cada componente.

CAPÍTULO 3

DISEÑO

En este apartado definiremos el diseño de nuestro sistema realizando un análisis de los componentes tecnológicos que necesitaremos para llevar a cabo nuestra plataforma de CI/CD cumpliendo con los requisitos establecidos. Después de identificar los diferentes componentes definiremos las relaciones entre cada una de ellos con el fin de obtener una estructura final que cumpla nuestras necesidades. Una vez dada la estructura final seleccionaremos un producto específico de cada componente según su ámbito tecnológico; Por ejemplo, en el caso del componente "programa", procederíamos a decidir que lenguaje de programación utilizaríamos.

Aunque el orden natural de definir la estructura parezca ser seleccionar directamente un producto específico de un componente nada mas identificarlo, es muy importante realizar la tarea de identificar los componentes y establecer una estructura sin especificar los productos en cuestión; Esto es debido a que algunos productos se acoplan mejor con otros, lo cual nos podría inducir a definir una estructura que depende de los productos seleccionados y no de nuestros requisitos.

3.1 Análisis de los componentes necesarios

A continuación analizaremos que componentes son necesarios para formar nuestra plataforma de CI/CD. Estos componentes tienen que poder satisfacer nuestros requisitos y llevar a cabo las acciones definidas en el ciclo de despliegue.

3.1.1 Programa

Para poder ver los resultados y cuantificar cuanto agilizamos el proceso de despliegue vamos a necesitar seleccionar un lenguaje de programación e implementar un pequeño programa con un conjunto de pruebas. Sobre este programa automatizaremos la supervisión de la calidad del código y su despliegue cada vez que apliquemos un cambio, formando el ciclo de despliegue.

Es importante saber que al igual que a la hora de programar los diferentes lenguajes se hace de manera diferente, el tratar su despliegue también es diferente. Las principales diferencias son la manera de ejecutar un programa según sea interpretado o compilado, y las instrucciones definidas en los clientes de cada lenguaje para realizar el empaquetado del software.

3.1.2 Sistema de control de código fuente

Una vez tengamos el código de un programa necesitaremos una tecnología para almacenar este software de manera segura sin perder agilidad a la hora de modificarlo, cumpliendo con los requisitos RF-2 (almacenar código), RF-3 (descargar código) y RF-4 (subir código).

Actualmente la tecnología predominante para llevar a cabo esta tarea es un sistema de control de código fuente (Source Code Management (SCM)), gracias a que cumple las buenas prácticas sobre gestión de la configuración del software establecidas en el ISO 12207 [5].

Un sistema de control de código fuente nos permite almacenar el código de nuestros programas en un repositorio. Para almacenar el código los usuarios tienen que subirlo desde su entorno local registrando dicha acción, de manera que en todo momento podemos saber qué cambios se han realizado y quien lo ha realizado.

3.1.3 Servidor de automatización

Lo siguiente que necesitaremos es un servidor de automatización que nos servirá como núcleo central para aportar la lógica necesaria para orquestar el flujo de CI/CD de nuestras aplicaciones.

Con una configuración adecuada podremos orquestar todas las etapas necesarias para poder poner el código en producción, entre ellas las definidas en los requisitos RF-7 (pruebas automáticas), RF-9 (generar artefacto) y RF-11 (desplegar un artefacto de código automáticamente).

La manera de configurar estas acciones dependerá del producto que seleccionemos para esta tecnología, pero en general las acciones se definen mediante lenguajes de consola como Bash de Unix o CMD de Windows.

3.1.4 Repositorio de artefactos

Seguidamente requeriremos de un repositorio de artefactos de tal manera que quede satisfecho el requisito RF-10 (Almacenar artefactos de código). Esto se traduce en una tecnología la cual nos permita almacenar los diferentes artefactos de las aplicaciones generados durante la fase de empaquetado.

Este repositorio debe permitir el almacenado de diferentes versiones de los artefactos generados para poder desplegar una versión u otra en cualquier momento sin tener que repetir la fase de empaquetado.

3.1.5 Servidor remoto

Finalmente para ejecutar los programas tras pasar por todo el ciclo de despliegue y que éstos sean visibles por el consumidor final tal como se requiere en el requisito RF-13 necesitaremos un servidor remoto. Este servidor remoto debe tener todas las dependencias instaladas y debe estar configurado para poder desplegar nuestros programas en él.

3.2 Interacciones entre los componentes

Una vez identificados los componentes hay que definir las relaciones e interacciones entre ellos con el fin de crear una estructura que forme nuestra plataforma de CI/CD.

3.2.1 Programa

El programa es el iniciador del ciclo de despliegue llevado a cabo por el sistema.

Este componente únicamente interactúa con el repositorio de código cuando un usuario integra un cambio en el repositorio del proyecto, lo que inicia todo el proceso de despliegue llevado a cabo por el sistema.

Una vez que el usuario sube el código al repositorio del programa únicamente tiene que esperar a que su desarrollo sea desplegado automáticamente para ver los cambios reflejados en el servidor remoto, gracias a la interacción entre el resto de componentes.

3.2.2 Sistema de control de código fuente

La entrada que recibe el sistema de control de código fuente es la acción de subir código a un repositorio llevada a cabo por un usuario. Una vez se realiza dicha acción, la salida es el inicio de un nuevo ciclo de despliegue sobre el código subido.

3.2.3 Servidor de automatización

El servidor de automatización es el componente que mas interacciones lleva a cabo ya que es el aquel que tiene la lógica para orquestar todo el ciclo de despliegue. Este componente realiza una serie de acciones secuencialmente y según sea su resultado lleva a cabo unas tareas u otras.

El evento inicial que recibe es la subida de código al repositorio. Con este evento la primera acción que lleva a cabo es descargar el código y posteriormente lanzar la batería de pruebas que éste lleva adheridas. Si las pruebas son satisfactorias el flujo

de despliegue continua mientras que en el caso contrario se detendría el ciclo de despliegue marcando esa versión de código como errónea.

3.2.4 Repositorio de artefactos

La siguiente acción después de realizar las pruebas es crear un artefacto con el código listo para ser ejecutado. Cuando el servidor de automatización acabada de empaquetar el código interacciona con el repositorio de binarios para almacenar la versión de código empaquetada.

Este artefacto almacenado tendrá un identificador de la versión para poder distinguirlo del resto de versiones del mismo programa.

3.2.5 Servidor remoto

Tras almacenar correctamente el artefacto con la versión del código que está pasando el ciclo de despliegue, el servidor de automatización se encarga de desplegar dicha versión en el servidor final para que se puedan ver los cambios reflejados. El artefacto de la versión lo tiene localmente el servidor de automatización desde la fase de empaquetado, por lo que no hará falta descargarnos dicha versión del repositorio de binarios para posteriormente desplegarla.

3.2.6 Estructura final

La siguiente imagen refleja de manera esquemática nuestra plataforma de despliegue con la interacción entre los componentes:

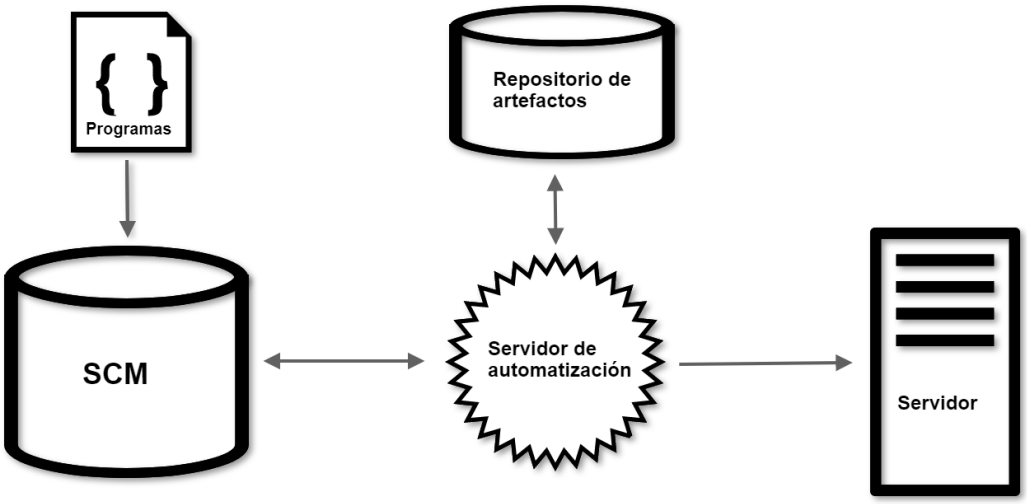

Figura 3.1: Estructura de nuestro sistema

3.3 Características de cada componente

A continuación vamos a analizar las características que deben ser cubiertas por el producto que seleccionemos de cada componente. Estas características nos aportan una serie de ventajas que nos facilitarán que la estructura de nuestro sistema cumpla los requisitos establecidos siendo fieles a la metodología DevOps.

3.3.1 Requisitos comunes

Únicamente se ha identificado un requisito que deben cumplir todas las herramientas que seleccionemos de cada componente y es que éstas sean gratuitas.

Componentes gratuitos

El primer requisito de nuestra plataforma de CI/CD es que todos los componentes que la forman sean gratuitos, con el fin de hacer que el sistema sea accesible a organizaciones con pocos recursos económicos.

3.3.2 Programas

Después de analizar las características comunes estudiaremos que características deben tener los programas y que sean fieles a la metodología DevOps.

Facilidad a la hora de detectar un cambio

El objetivo de este trabajo de fin de grado no es elaborar un desarrollo de software, por lo que los programas que llevaremos a cabo serán programas muy simples sobre los cuales sea muy sencillo identificar los incrementales o las nuevas versiones que se integren en el repositorio.

Orientación a microservicios

Las aplicaciones que llevaremos a cabo y los lenguajes que seleccionamos deben estar orientados a microservicios. “El término de microservicio hace referencia a un ecosistema de aplicaciones mínimas que interaccionan entre ellas en contra de la construcción de un gran programa monolítico"[7].

Este es un requisito de nuestro trabajo porque en un entorno de entrega continua es recomendable tener bien separadas las responsabilidades de nuestra solución en diferentes servicios con el fin de tener bien focalizados los cambios que subimos.

Servicio web

Para poder verificar que las versiones generadas por los cambios generados en el código se despliegan correctamente trabajaremos únicamente con servicios web. De esta manera podremos revisar si nuestra aplicación ha desplegado la nueva versión correctamente únicamente consultando el navegador apuntando al servidor remoto.

3.3.3 Sistema de control de código fuente

El sistema de control de código fuente que seleccionemos debe tener las siguientes características:

Permitir el trabajo simultaneo

Nuestro SCM debe permitir que trabajen múltiples usuarios sobre un mismo repositorio.

Agilidad al integrar cambios

Para poder trabajar siguiendo una metodología DevOps necesitaremos que el trabajo de integrar nuestros cambios al SCM no suponga ningún inconveniente y se debe poder realizar de la manera más segura y rápida. Esto se debe a que siguiendo ésta metodología los cambios que se realizan tienen un contenido mínimo pero son muy frecuentes, a diferencia de un modelo waterfall donde se hacen subidas con mucho contenido pero con poca frecuencia.

3.3.4 Servidor de automatización

Seguiremos el análisis de las características de los componentes analizando las del servidor de automatización.

Capacidad de integración con diferentes tecnologías

El servidor de automatización es la pieza que se encargará de integrar todos los componentes de nuestra plataforma de CI/CD. Por este motivo se necesita que esta herramienta sea capaz de integrarse con diferentes tecnologías.

Interfaz gráfica de usuario

El servidor de automatización es el encargado de controlar los eventos y acciones que están ocurriendo en nuestra plataforma. Mediante una interfaz gráfica de usuario los desarrolladores podrán visualizar el estado de sus ciclos de despliegue e incluso las acciones que se están llevando a cabo en un momento determinado.

Configuración accesible a nivel de proyecto

Para mantenernos fieles a la metodología DevOps esta herramienta tiene que permitir configurar los ciclos de CI/CD a nivel de proyecto. Esto quiere decir que el desarrollador que realiza la aplicación tiene que poder permitir configurar su flujo de despliegue y tomar decisiones sobre éste. Por ejemplo debe tener la capacidad de decidir si ejecutar o no los tests funcionales sobre el código.

Es importante que se pueda diferenciar la configuración global del servidor de automatización de la configuración del proyecto para evitar que un error llevado a cabo por una persona impacte en los demás.

3.3.5 Repositorio de artefactos

A continuación analizaremos que características debería tener el repositorio de artefactos para hacer una selección óptima una herramienta para este componente.

Versionado

El repositorio de artefactos debe de mantener el versionado de los distintos paquetes de código, de manera que no sea necesario pasar por todo el flujo de integración continua otra vez si queremos revertir el despliegue de una versión.

API

Una API es “un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. API significa interfaz de programación de aplicaciones"[8]. Mediante una API nos aseguramos que se puedan automatizar las funciones del repositorio mediante código.

3.3.6 Servidor remoto

Finalmente analizaremos las características que tiene que tener el servidor donde se van a ejecutar nuestros programas.

Servidor Operativo con bajo consumo de recursos

Como vamos a utilizar el servidor como servidor web no necesitamos un sistema operativo diseñado para uso doméstico. De esta manera nos ahorraremos todos los recursos que se invierten en facilidades para el usuario como la interfaz gráfica de usuario.

Conectividad a la red pública

El servidor remoto tiene que tener conectividad a la red pública para que los consumidores de los programas de la organización puedan acceder a ellos.

En esta sección hemos analizado las características que debe tener cada componente. En el siguiente apartado realizaremos la selección de una herramienta específica para cada uno de ellos basándonos en las características analizadas anteriormente.

3.4 Selección de los componentes

A continuación vamos a seleccionar un producto específico para cada componente. Para seleccionar el producto óptimo tendremos en cuenta los requisitos de nuestro sistema, las características que esperamos de cada componente y la comunidad que envuelve al producto.

3.4.1 Programa

La finalidad de este trabajo es hacer una plataforma que acompañe al ciclo de despliegue de un proyecto software por lo que cualquier lenguaje que seleccionemos se

adaptará a nuestros requisitos ya que todos tienen su manera de llevar a cabo las fases de test, empaquetado y despliegue.

Los lenguajes candidatos son los siguientes, basándonos en el requisito de poder ver contenido en una web:

- Python.
- Go.
- Java.
- Javascript.
- PHP.
- C.
- C++.
- C Sharp (C#).
- Visual Basic (VB).

Uno de los primeros criterios que miraremos para seleccionar un lenguaje de programación es que el lenguaje pueda tener un servidor web embebido de tal manera que éste sea instalado junto a la aplicación. Este criterio también aportará la fortaleza de poder configurar ciertos parámetros de dicho servidor desde el código de la aplicación, evitando así la necesidad de conocer los sistemas donde éste está instalado.

Esta característica es favorable a la metodología DevOps ya que la misma persona que desarrolla la aplicación se puede encargar de configurar y gestionar el servidor web desde la propia aplicación sin la necesidad de poseer conocimientos técnicos sobre administración de sistemas.

Basándonos en esta característica nos quedan los lenguajes GO, Java junto al marco de trabajo Spring Boot [9], PHP, C# con el marco de trabajo .NET [10] y VB también con el marco de trabajo .NET. En el caso de PHP y de Python nos avisan que los servidores web integrados que se incluyen no están preparados para ejecutar cargas de trabajos en producción, por lo que quedan descartados [11, 12].

Finalmente seleccionamos el lenguaje de Java junto al marco de trabajo de Spring Boot, descartando C#, GO y VB basándonos en la popularidad de los lenguajes seleccionados [13]. En la siguiente figura podemos ver una tabla con la popularidad de los distintos lenguajes en Julio de 2019 respecto al año anterior, basándose en las búsquedas realizadas en los principales motores de búsqueda [14].

Worldwide, Jul 2019 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	28.24 %	+4.4 %
2		Java	19.99 %	-2.1 %
3		Javascript	8.55 %	+0.1 %
4	↑	C#	7.43 %	-0.5 %
5	↓	PHP	6.92 %	-1.1 %
6		C/C++	5.99 %	-0.1 %
7		R	4.14 %	+0.0 %
8		Objective-C	2.82 %	-0.6 %
9		Swift	2.52 %	-0.3 %
10		Matlab	1.85 %	-0.4 %
11	↑	TypeScript	1.72 %	+0.2 %
12	↓	Ruby	1.42 %	-0.2 %
13		VBA	1.41 %	-0.0 %
14	↑↑	Kotlin	1.41 %	+0.5 %
15	↑↑	Go	1.17 %	+0.3 %

Figura 3.2: Popularidad de los lenguajes [14].

3.4.2 Sistema de control de código fuente

El siguiente componente a seleccionar es un Sistema de control de código fuente donde almacenamos los programas de software realizados.

Existen dos tipos de sistemas de control de código entre los cuales tendremos que elegir la solución más adecuada a nuestros objetivos: los centralizados y los distribuidos. Los sistemas de control centralizados son aquellos en los cuales únicamente existe un repositorio con todo el código y el historial de cambios mientras que en un sistema de control distribuido encontramos un conjunto de repositorios locales que se sincronizan con un repositorio central. En las figuras 3.2 y 3.3 podemos ver de manera esquemática las diferencias entre un sistema de control de versiones distribuido y centralizado.

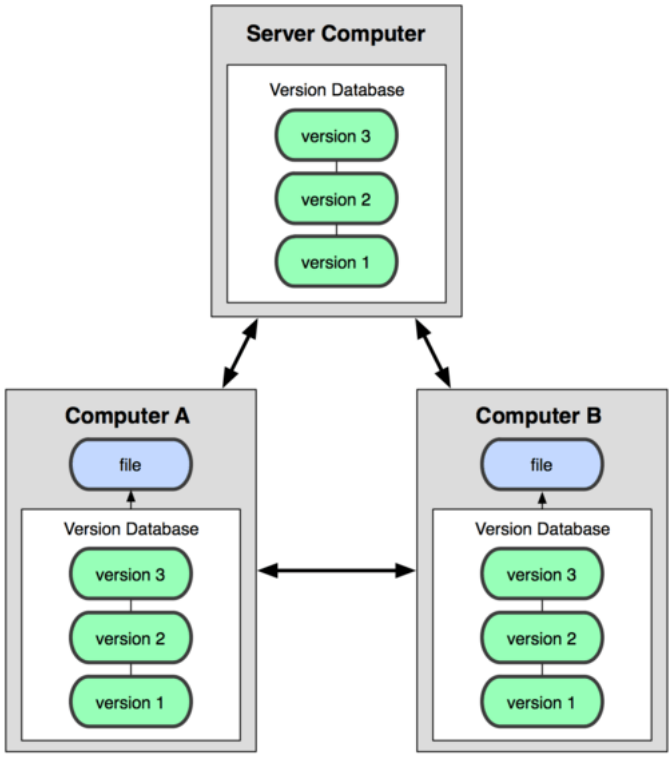

Figura 3.3: Sistema de control distribuido [15]

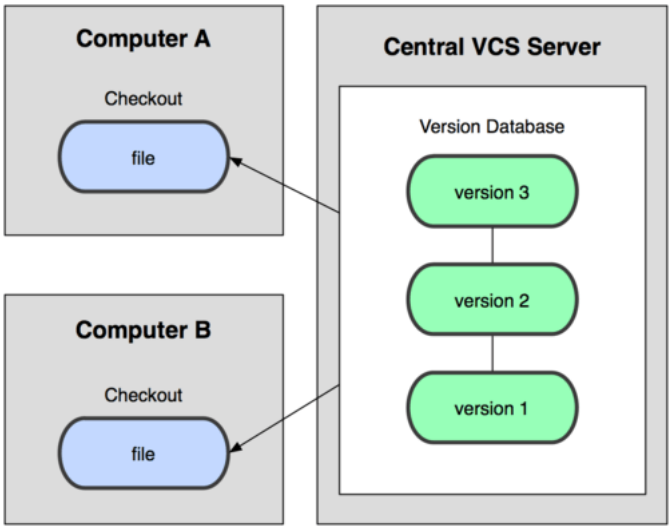

Figura 3.4: Sistema de control centralizado [15]

Las ventajas y desventajas de utilizar un sistema de control de versiones distribuido contra uno centralizado son las siguientes [16]:
Ventajas:

- Realizar acciones que no sean subir o descargar código son extremadamente rápidas ya que no necesitan acceder a un repositorio remoto.
- Se pueden integrar cambios en una rama local y posteriormente subir al repositorio todos los cambios integrados en dicha rama.
- Al poder interactuar con el repositorio en local se puede trabajar sin internet.
- Al tener cada programador una copia completa del proyecto, éste es capaz de compartir sus cambios locales con otros programadores antes de integrar sus cambios en el repositorio.

Desventajas:

- Si el proyecto contiene archivos binarios pesados el espacio necesario para guardar el histórico de cambios de esos ficheros puede crecer en exceso.
- La descarga de un proyecto con un histórico muy grande puede necesitar una cantidad de espacio en disco y tiempo de descarga elevados.

En nuestro caso trabajaremos con un software de control de versiones distribuido ya que la agilidad que nos aporta únicamente se ve mermada al trabajar con proyectos muy extensos o con binarios. En nuestro caso si nos adherimos a la metodología DevOps trabajaremos con programas pequeños orientados a los microservicios, por lo que no estaremos expuestos a las desventajas que nos aporta este tipo de software de control de versiones.

Basándonos en algunas estadísticas extraídas de Google trends [17], la solución mas popular de este tipo de software es git, comparado con sus competidores mas potenciales como Mercurial o Plasticscm. Dentro de esta tecnología han aparecido diferentes proyectos para dotarla de mas funcionalidad como una interfaz de usuario, soporte para wikis o medidas de seguridad sobre los repositorios. En nuestro proyecto seleccionaremos uno de estos proyectos para aprovecharnos de sus mejoras en la funcionalidad.

En nuestro caso utilizaremos la solución aportada por GitHub, una plataforma de desarrollo colaborativo donde se alojan una gran multitud de proyectos abiertos en los cuales se permite la colaboración de usuarios registrados. La selección de esta solución se debe a la gran comunidad que hay detrás de esta plataforma en la cual se alojan grandes proyectos de código abierto con una gran cantidad de colaboradores [18]. Además en nuestro caso nos es beneficioso publicar los programas que utilizaremos y nuestra plataforma de CI/CD para darles visibilidad a los interesados en ello.

Si fuéramos una empresa a la cual no le interesa publicar sus proyectos se barajarían soluciones que nos permitan instalar la plataforma en un servidor local de manera gratuita como por ejemplo GitLab. De este modo nuestro sistema de control de código fuente sería gestionado internamente.

3.4.3 Servidor de automatización

El servidor de automatización es el componente que mas acciones llevará a cabo en nuestra plataforma. Muchas soluciones de esta herramienta han sido específicamente creadas para dar soporte a la integración continua y al despliegue continuo. Para seleccionar esta herramienta nos centraremos en los requisitos de capacidad de integración, interfaz de usuario y la configuración accesible a nivel de proyecto. Actualmente las soluciones de servidores de automatización basadas en código libre mas utilizadas en el mercado son las siguientes:

- TeamCity.
- Circle CI.
- Travis CI.
- Jenkins.
- GoCD.
- Gitlab CI.

De la lista anterior descartaremos directamente los servicios con una capa gratuita con limitaciones, como TeamCity, CircleCI o Gitlab CI en los cuales nos limitan el número de configuraciones, las ejecuciones en paralelo o incluso funcionalidades.

De las soluciones restantes utilizaremos Jenkins puesto que es la mas sólida debido a que cuenta con una comunidad muy extensa y activa, lo cual implica que este proyecto seguirá actualizándose y evolucionando.

La gran popularidad de esta herramienta en el sector ha obligado a grandes compañías tecnológicas a facilitar que sus productos orientados a CI/CD se puedan integrar con Jenkins [19]. En cuanto a su interfaz gráfica es de las mas pobres, pero cumple con creces toda la funcionalidad requerida por nuestra plataforma de CI/CD de visualizar todas las etapas del ciclo de despliegue de una manera intuitiva.

Esta herramienta posee una gran capacidad de integrarse con otras tecnologías gracias a las extensiones creadas por la comunidad y a su vez también nos permite realizar una configuración de nuestro ciclo de despliegue adaptada a nuestras necesidades.

La manera de trabajar con Jenkins es abstraer un conjunto de acciones automatizadas en un término llamado 'pipeline'. Un pipeline es un conjunto de acciones que se llevan a cabo con un determinado contexto. A modo de ejemplo, en caso de poseer diferentes programas con el mismo lenguaje se crearía un pipeline para cada programa. Ambos pipelines realizarían las mismas acciones pero bajo un contexto diferente. De esta manera cada aplicación tendrá su ciclo de despliegue con su respectivo estado. En nuestro caso un pipeline será el conjunto de acciones que forma el ciclo de despliegue de un proyecto:

- Descarga de código del repositorio.
- Lanzamiento de la batería de pruebas.
- Construcción y almacenamiento del artefacto.

- Subida del artefacto al repositorio de artefactos.

3.4.4 Repositorio de artefactos

El repositorio de artefactos es aquel componente cuya finalidad es almacenar los artefactos de los programas que han pasado satisfactoriamente la batería de pruebas. Estos programas se almacenarán listos para ser descargados en un servidor remoto en forma de artefacto, el cual contiene todo lo necesario para para ser ejecutadas.

Los requerimientos de este componente son muy simples, únicamente se requiere que los artefactos estén versionados para ser ágiles a la hora de volver a atrás un despliegue y que exista una API que nos permita al resto de componentes interactuar con nuestro repositorio de código.

Las soluciones mas completas que podemos encontrar de esta herramienta son las siguientes:

- Nexus OOS.
- JFrog artifactory.
- Repositorio git.
- Almacenamiento en la nube.

Para empezar descartaremos la opción de pago de almacenamiento en la nube, pero queremos hacer una mención especial a esta herramienta. El almacenamiento en la nube es de pago por lo que no cumple los requisitos de este TFG, pero para un entorno empresarial es una opción muy válida ya que el almacenamiento en disco es relativamente barato y nos aporta una gran escalabilidad ya que los recursos para llevar a cabo este almacenamiento son dinámicos, lo cual nos libera de las tareas de mantenimiento.

La siguiente solución a tener en cuenta es un repositorio git. Esta tecnología nos otorga un versionado de todo lo que modificamos en un repositorio. De esta manera podemos almacenar en un repositorio el artefacto a desplegar y descargarlo en el servidor final haciendo simplemente un "git pull", que es la instrucción con la cual nos descargarnos el contenido de un repositorio.

En caso de querer una versión antigua de un artefacto únicamente tendríamos que descargarnos la versión anterior del repositorio, lo cual es posible ya que git mantiene un versionado de lo que vamos subiendo. Esta opción es muy práctica, sin embargo nos puede causar problemas ya que esta tecnología no está pensada para ser utilizada de este modo. Las malas prácticas nos conllevarían inconvenientes y labores de mantenimiento adicionales, por lo cual descartaremos esta solución. Recordemos que si almacenamos binarios en git el espacio que ocupa en disco el histórico puede aumentar drásticamente obligándonos a tener que realizar restauraciones periódicas del repositorio para reducir dicho espacio.

Las dos soluciones que nos quedan para seleccionar son las mas adecuadas para llevar a cabo la función de repositorio de artefactos ya que han sido diseñadas para ese fin. Ambas soluciones también son utilizadas como repositorio de dependencias facilitándonos el uso de librerías propias en nuestro programa, sin embargo el uso que le daremos a esta herramienta va a ser puramente para almacenar binarios.

En esta ocasión seleccionaremos como solución de repositorio de artefactos Artifactory de JFrog debido que su API es muy intuitiva [20] y además incluye hasta un cliente con el cual podemos manipular los artefactos. Hay que destacar que esta herramienta es menos flexible que Nexus, el cual permite añadir código a los repositorios como por ejemplo una tarea programada para eliminar artefactos antiguos, sin embargo para este componente no requerimos de mucha flexibilidad ya que únicamente lo utilizaremos para almacenar binarios.

3.4.5 Servidor remoto

Finalmente nos falta decidir en que servidor queremos desplegar nuestras aplicaciones. El requisito principal sobre esta tecnología es que el servidor sea ligero puesto que únicamente va a ser utilizado para servir el contenido de nuestra aplicación web. No es necesario malgastar recursos en facilidades de uso como una interfaz de usuario o aplicaciones preinstaladas.

La primera decisión que debemos tomar para seleccionar una solución de este componente es elegir entre las dos grandes familias de sistemas operativos: Windows o Linux. En nuestro caso seleccionaremos Linux puesto que los recursos consumidos por este sistema al ejecutar programas en java son inferiores a Windows[21]. También seleccionaremos un Sistema Operativo (SO) basado en esta familia puesto que al ser de software libre su compatibilidad con otros componentes es mayor, a causa de la gran comunidad que le da soporte.

Dentro de los SO basados en Linux seleccionaremos uno basado en Debian puesto que su sistema de paquetería Advanced Packaging Tool (APT) es generalmente mas conocido que su mayor competidor Red Hat Package Manager (RPM), aunque ambas opciones podrían ser perfectamente válidas. Basados en Debian encontramos principalmente dos sistemas operativos: el propio Debian y Ubuntu.

En nuestro caso seleccionaremos Debian en su versión de servidor ya que este sistema operativo viene con una configuración mínima y una base estable. Esta configuración mínima hace que los recursos del SO no sean desaprovechados en aspectos que no son servir nuestra aplicación, lo cual es muy importante en un entorno de producción.

3.4.6 Resultado

El resultado de la selección de las soluciones para cada tecnología es la siguiente:

- **Java** como lenguaje de programación para nuestro programa.
- **GitHub** como repositorio de código.
- **Jenkins** como servidor de automatización.
- **Artifactory** como repositorio de binarios.
- **Debian** como sistema operativo del servidor remoto.

Y de manera mas ilustrativa:

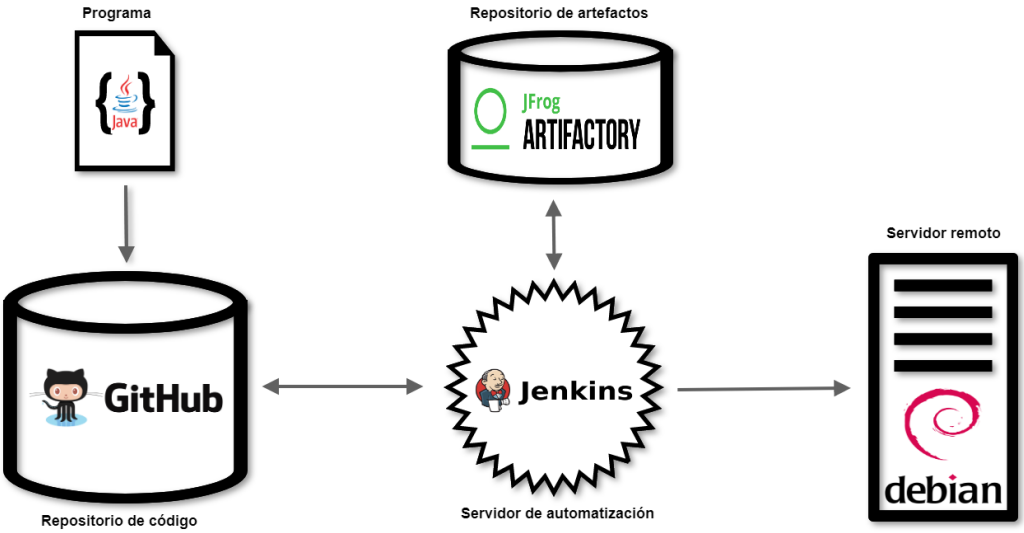

Figura 3.5: Estructura de las herramientas utilizadas en nuestro sistema.

Una vez seleccionadas las soluciones para cada tecnología empezaremos con la parte de la implementación, donde empezaremos a instalar y configurar los componentes para materializar nuestra plataforma de CI/CD.

CAPÍTULO 4

IMPLEMENTACIÓN

El objetivo de este apartado es realizar la implementación de nuestro sistema. Instalaremos los componentes y los configuraremos adecuadamente para que se lleve a cabo el ciclo de despliegue esperado. Toda esta instalación y configuración de los componentes se ha realizado sobre un servidor local, puesto que en este TFG queremos destacar el proceso funcional aportado por el sistema y no la infraestructura sobre la cual éste se sustenta.

Una vez instalados y configurados los componentes explicaremos para cada componente la funcionalidad que llevan a cabo durante el ciclo de despliegue.

4.1 Programa

Como se ha especificado anteriormente el programa lo utilizaremos únicamente como objeto para mostrar el resultado del ciclo de despliegue, integrando automáticamente los cambios en nuestro servidor con la aplicación ejecutándose. Por este mismo motivo el programa es muy sencillo. El código lo podemos encontrar en el siguiente repositorio: https://github.com/sergimrz-tfg/dummy-service-java.

El programa en java realizado lleva a cabo la función de simular la gestión básica del versionado de una aplicación. Esta versión está formada por los atributos MAJOR, MINOR y PATH, donde únicamente PATCH no es final por lo cual es modificable. De este modo para subir la versión MAJOR o MINOR tendremos que realizar un despliegue nuevo con lo cual lanzaremos nuestro ciclo de despliegue. Las acciones que puede llevar a cabo el programa son:

- **Consultar la versión:** mediante la ruta /version podremos visualizar la cadena de texto "You are running the version $VERSION.[en] el navegador.
- **Incrementar el PATCH de la versión:** accediendo a la ruta /upgrade incrementaremos en uno el PATCH de la versión, devolviendo la cadena "Successfully

upgraded the version to $VERSION+1".

En este programa hay que recalcar el uso del marco de trabajo SpringBoot. Este marco de trabajo nos permite utilizar anotaciones Java[22] para facilitar la programación. Esto se refleja en nuestro código a la hora de mapear métodos a las rutas establecidas.

Mediante estas anotaciones nuestra aplicación es capaz de levantar un servidor web Tomcat donde se expone ella misma. Este servidor web se puede configurar desde la propia aplicación, y en nuestro caso lo hemos establecido bajo el puerto 8080. Esto es un buen ejemplo de una práctica propia de la metodología DevOps, ya que desde el código podemos configurar recursos que se consideran de sistemas.

En el mismo repositorio del programa también podemos encontrar los tests que hemos definido para realizar las pruebas. Estos tests son dos pruebas que comprueban el funcionamiento del programa principal:

- La primera prueba revisa que el texto que se muestra al arrancar la aplicación es correcto.
- La segunda prueba comprueba que al arrancar la aplicación si llama a la ruta de /upgrade se incremente la versión mostrada por pantalla.

Para realizar la compilación, los tests, y el empaquetado del código utilizaremos la herramienta Maven, que nos facilita la construcción de proyectos Java [23].

4.2 Repositorio de código

Para tener organizado todo lo que concierne al TFG en un mismo espacio se ha creado una organización de GitHub. Según GitHub las organizaciones son “cuentas compartidas donde las empresas y los proyectos de código abierto pueden colaborar en muchos proyectos a la vez. Los propietarios y administradores pueden administrar el acceso de los miembros a los datos y proyectos de la organización con funciones administrativas y de seguridad sofisticadas"[24].

Dicha organización la podemos encontrar en https://github.com/sergimrz-tfg, donde encontramos los repositorios con el código utilizado en este TFG.

4.3 Servidor de automatización

El servidor de automatización es el componente que realiza la lógica para llevar a cabo todo el ciclo de despliegue. En nuestro caso, hemos seleccionado Jenkins como solución tecnológica para este componente. Con esta herramienta diseñaremos un método para que los desarrolladores puedan gestionar su propio despliegue, ya que en la filosofía DevOps la misma persona que realiza un cambio es responsable de su puesta en producción.

Los ciclos de despliegue en Jenkins se llevan a cabo en forma de pipelines, tal como se ha especificado en la etapa de diseño. Para dotar al desarrollador de la capacidad de gestionar los pipelines utilizaremos otra característica de Jenkins que nos permite configurarlos mediante código almacenado en un fichero comúnmente llamado Jenkinsfile.

De no utilizar esta característica cada pipeline se tendría que configurar desde interfaz gráfica de la herramienta, mientras que mediante el uso del fichero Jenkinsfile la configuración de los pipelines quedará repositada junto al proyecto, tal como se puede ver en el siguiente fichero de nuestro programa en Java: https://github.com/sergimrz-tfg/dummy-service-java/blob/master/Jenkinsfile.

Las acciones que se llevan a cabo en los pipelines para realizar las diferentes fases del ciclo de despliegue son generalmente llamadas a la consola del sistema operativo donde esté instalado el servidor de automatización. Los lenguajes de consola no son generalmente conocidos, por lo que para reducir la curva de aprendizaje necesaria para que un desarrollador pueda configurar sus propios pipelines programaremos unas librerías en Groovy llamadas librerías compartidas[25] que nos abstraigan de los comandos de consola.

Adicionalmente a facilitar la configuración de los pipelines a los desarrolladores, realizar la abstracción de los comandos en una librería compartida también nos beneficia a la hora de realizar cambios en los comandos que llevan a cabo las acciones, ya que de esta manera tenemos de forma centralizada la definición de estas instrucciones. Mediante estas librerías podemos modificar las acciones que se llevan a cabo en los pipelines únicamente modificando el repositorio de la librería compartida; si no realizáramos esta abstracción y tuviéramos los comandos definidos en los Jenkinsfile, realizar un cambio de una acción implicaría cambiar todos los proyectos de la organización modificando sus Jenkinsfile.

Estas librerías se pueden encontrar en el siguiente repositorio: https://github.com/sergimrz-tfg/shared-library.

Para iniciar el pipeline automáticamente cada vez que se realiza un cambio en el repositorio de código se ha configurado en el Jenkinsfile un disparador que realiza revisiones periódicas al repositorio cada minuto. Si se detecta un cambio de código el pipeline se iniciará, mientras que de lo contrario no se iniciará ninguna ejecución.

Hay maneras mas eficientes para detectar los cambios como realizar una llamada HTTP desde el repositorio de código al Jenkins cada vez que se detecta un evento de subida de código, de manera que nos evitamos realizar sondeos del repositorio cada minuto. Sin embargo, esto requiere que la red donde se ejecuta nuestro Jenkins sea accesible desde la red pública, lo cual no esta contemplado en el diseño de la infraestructura de nuestro sistema.

4.4 Repositorio de artefactos

Como solución para el repositorio de artefactos hemos seleccionado Artifactory[26]. Esta herramienta nos ofrece un software para la gestión y almacenamiento de los artefactos generados durante el ciclo de despliegue. El objetivo de esta herramienta es almacenar un conjunto de artefactos de cada programa para facilitar los despliegues. Gracias a tener almacenados los artefactos generados no necesitamos compilar la misma versión de código cada vez que queramos desplegar dicha versión. Algunas acciones que nos demuestran la flexibilidad que nos aporta tener los artefactos repositados son:

- Compilar una versión y desplegarla posteriormente bajo demanda.
- En caso de que nuestra plataforma admita auto escalado, desplegar la nueva instancia con una de las versiones almacenadas.
- Revertir una versión.
- Cambiar el servidor final desplegando una versión ya almacenada.

Para una óptima organización de los artefactos, crearemos un repositorio de artefactos por proyecto. De esta manera cada repositorio contendrá un conjunto de versiones de un único proyecto. Para versionar los proyectos que se suben al repositorio utilizaremos el número de la ejecución del pipeline de Jenkins que ha generado esa versión, de manera que cada vez que se realice un cambio se incrementará dicha versión, tal como se muestra en la figura. El versionado del proyecto también se puede realizar desde la configuración del proyecto en caso de que se desee llevar a cabo un versionado mas manual desde la configuración del proyecto, pero en este trabajo nos hemos decantado por automatizar el versionado de los proyectos. La siguiente imagen muestra como quedan os artefactos versionados en la herramienta:

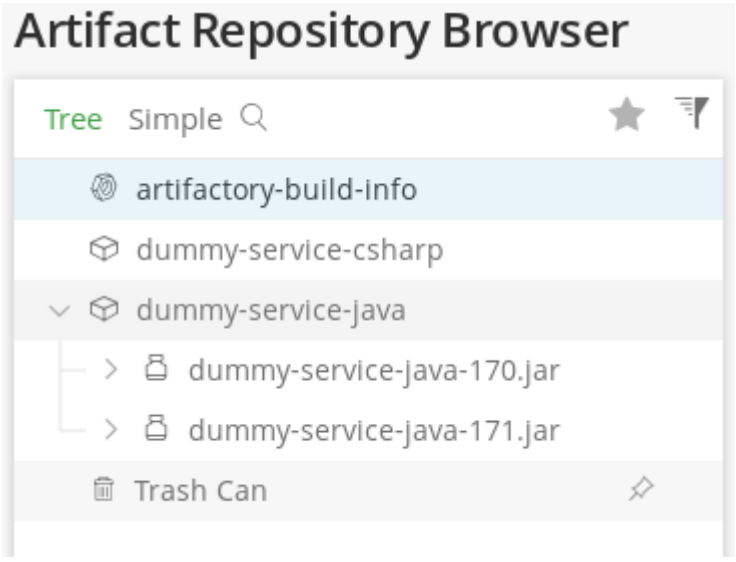

Figura 4.1: Navegador del artifactory con artefactos versionados

La subida del artefacto se realiza desde el Jenkins cuando se llega a la etapa correspondiente. Esto se hace mediante un curl proporcionado por la documentación de Artifactory, al cual se le indica el host de Artifactory que en nuestro caso sera localhost y el archivo a subir en el sistema de ficheros. Este curl lo abstraeremos en nuestra librería para que el desarrollador solo tenga que llamar a una función en groovy para realizar la subida del artefacto.

4.5 Servidor remoto

Para dar mas sensación de autenticidad al trabajo hemos optado por utilizar un servidor remoto ofrecido por el proveedor de servicios Google Cloud, de manera que la instalación del sistema operativo nos viene dada ya que únicamente hemos tenido que seleccionar la imagen base de Debian, que es el sistema operativo seleccionado en la fase de diseño. Para poder llevar a cabo el todas las acciones del ciclo de despliegue correctamente, se han realizado una serie de configuraciones en el servidor:

- Se ha expuesto el puerto 8080 del servidor con el fin de publicar a internet nuestro programa, modificando las reglas del cortafuegos incluidas en la configuración de Google Cloud Platform.
- Se han añadido las claves SSH del usuario de la instalación de Jenkins para poder lanzar comandos remotos desde el servidor donde está instalada esta herramienta. Estas claves son necesarias para que el servidor donde está instalado el sistema pueda interactuar con el servidor remoto lanzando comandos en él.
- Se han instalado las dependencias para poder ejecutar programas escritos en Java.
- Se ha creado un servicio del sistema que hace referencia a nuestro programa, el cual se ejecutará en segundo plano.

Para la parte de realizar el despliegue del artefacto en el servidor utilizaremos el comando Secure Copy (SCP)[27] que se encarga de copiar archivos desde un servidor local hasta un servidor destino. De este modo al llegar el pipeline a la fase de despliegue, Jenkins utiliza este comando para copiar el fichero del binario generado durante el ciclo de despliegue a nuestro servidor remoto.

Para ejecutar el binario una vez copiado en el servidor remoto operaremos el servicio del programa creado anteriormente, el cual lo ejecuta mediante el cliente de Java. De esta manera Jenkins se encarga de iniciar el servicio una vez copiado el artefacto mediante SSH.

CAPÍTULO 5

RESULTADOS

Una vez implementada nuestra plataforma de CI/CD vamos a simular realizar una serie de cambios, asegurándonos de que cubrimos todos los casos posibles de nuestro sistema. También analizaremos si nuestro sistema cumple con los requisitos establecidos en el inicio del proyecto.

5.1 Funcionamiento de la plataforma

En este apartado simularemos los casos de uso que intervienen con el sistema para ver su funcionamiento. Concretamente simularemos los casos de uso de integrar un cambio correcto, y otro con un cambio fallido; Para cada caso de uso veremos la perspectiva del usuario que realiza el cambio.

5.1.1 Cambio satisfactorio

El funcionamiento de la plataforma desde la perspectiva de un usuario es realmente sencilla. Partiendo desde un estado inicial, nuestro código muestra la versión 1.0.0 cuando hacemos una llamada a la ruta /version, tal como aparece en la siguiente imagen:

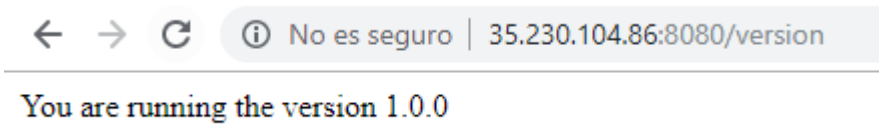

Figura 5.1: Estado inicial de nuestra aplicación.

La acción que desencadena todo el proceso llevado a cabo por el sistema es la integración de un cambio en el repositorio del código. En este caso de uso realizaremos una subida de código al repositorio del proyecto con un cambio que aumenta el campo

menor de la versión, que representa el segundo número de la cadena de versión. El cambio que se ha llevado a cabo es el que se muestra en la imagen:

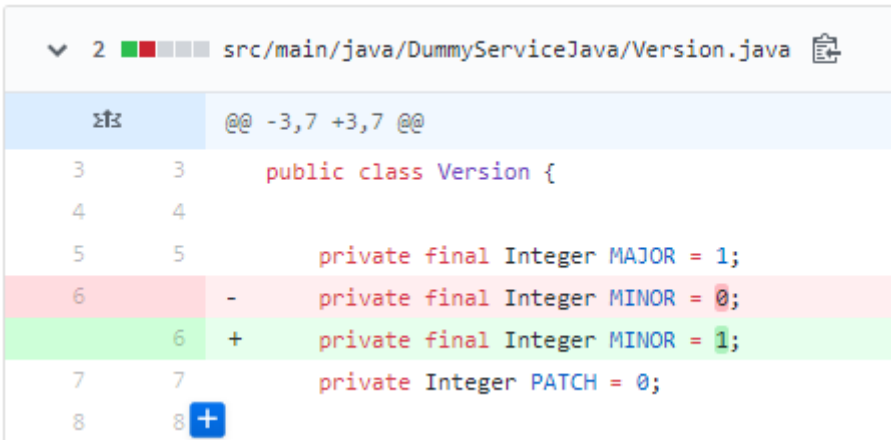

Figura 5.2: Cambio realizado en la aplicación.

Desde la perspectiva del usuario, al cabo de un minuto y treinta segundos este usuario puede ver ejecutándose su cambio en el servidor final, viéndose el resultado de la imagen:

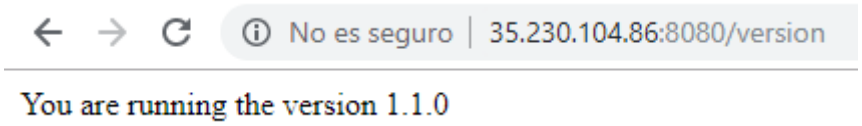

Figura 5.3: Cambio realizado en la aplicación.

Desde la perspectiva del sistema, durante este minuto y medio se ha llevado a cabo toda la fase de despliegue del programa, llevando a cabo todas las etapas del ciclo de despliegue:

- Descarga del código.
- Test.
- Empaquetado.
- Almacenamiento del artefacto.
- Despliegue en servidor final.

Este proceso se puede visualizar en el servidor de automatización, en nuestro caso Jenkins, y se muestran todas las etapas como satisfactorias:

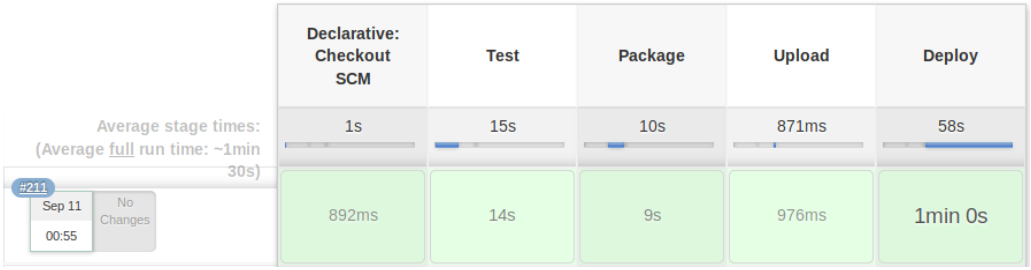

Figura 5.4: Cambio realizado en la aplicación.

5.1.2 Cambio insatisfactorio

Para simular un cambio insatisfactorio realizaremos un cambio para que la ruta /upgrade no incremente la versión, tal como se muestra en la siguiente imagen. Este cambio hará que fallen los tests, ya que un test comprueba que se incrementa la versión cuando accedemos a la ruta /upgrade:

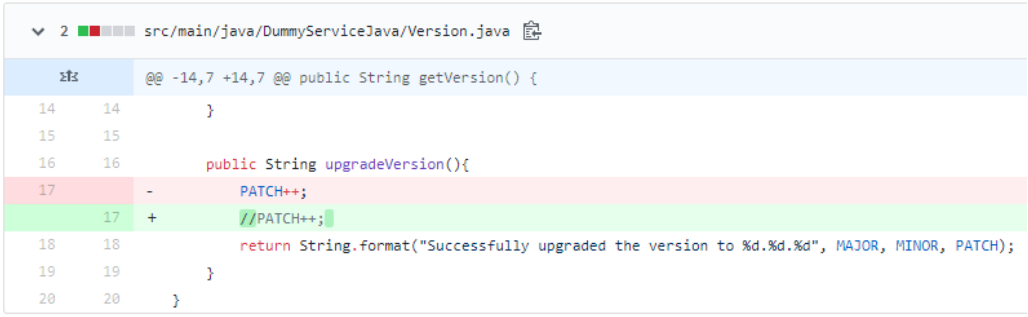

Figura 5.5: Cambio realizado en la aplicación forzar un test fallido.

En este caso pasado un tiempo prudencial el desarrollador notará que el cambio realizado no se ha desplegado, por lo que éste consultará la interfaz gráfica de Jenkins para revisar el pipeline, el cual al fallar se mostrará de la siguiente manera:

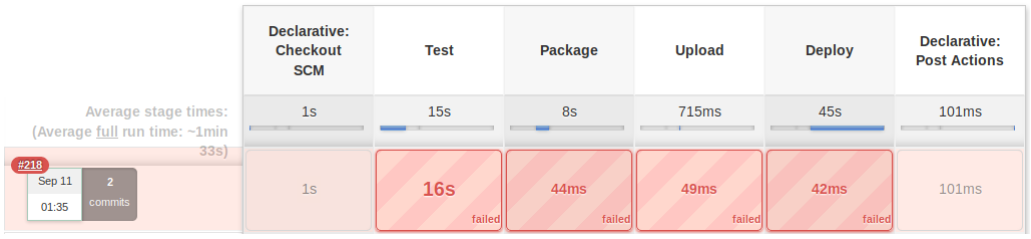

Figura 5.6: Pipeline fallido.

Eel desarrollador puede consultar el resultado de los tests con el fin de encontrar donde está el error, lo cual se muestra del siguiente modo:

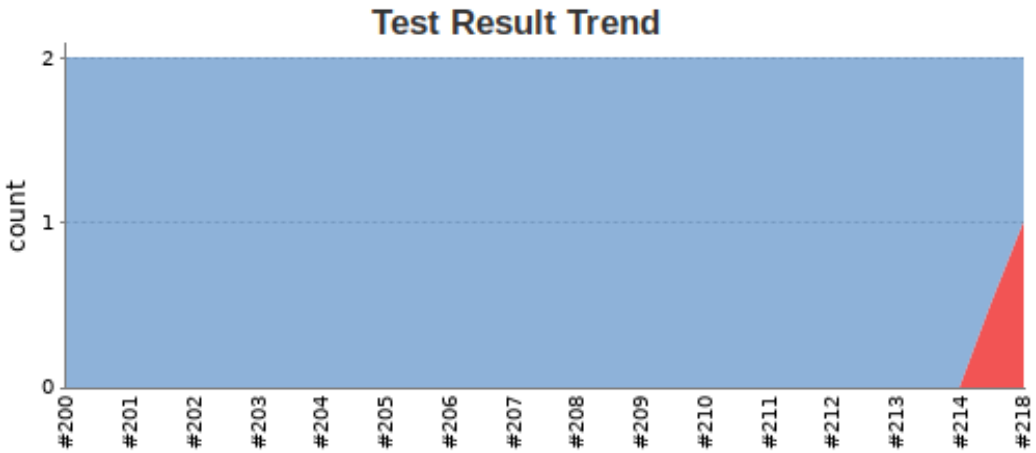

Figura 5.7: Gráfica con tendencia de los tests.

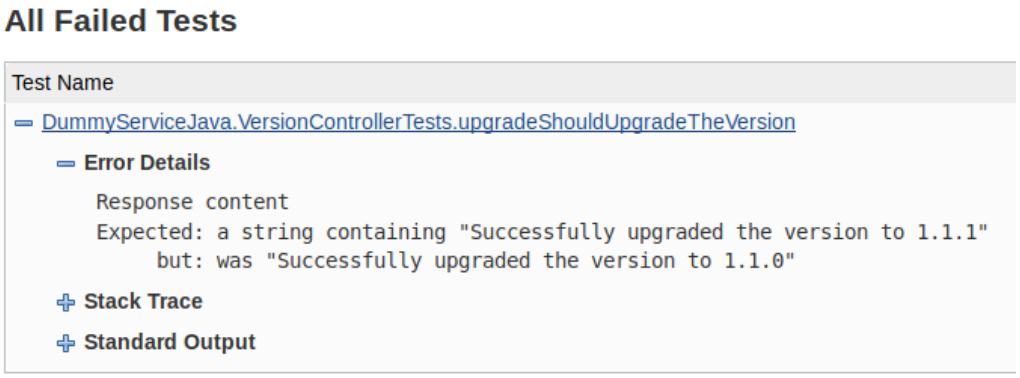

Figura 5.8: Resultado de los tests.

En este punto, el desarrollador se daría cuenta del error y prepararía una nueva versión válida. En este segundo caso el sistema únicamente ha llevado a cabo la fase de test, por lo que esta versión fallida ni se almacenaría en el repositorio de artefactos ni se desplegaría.

5.2 Aceptación de los requisitos

A continuación revisaremos que todos los requisitos definidos inicialmente son cubiertos por nuestra plataforma de CI/CD.

5.2.1 RF-1. Loguearse en el sistema

Todos los componentes que hemos seleccionado tienen de manera nativa su propio sistema de gestión de usuario, por lo que permitir que un usuario se autentique en el conjunto del sistema es posible dando de alta el usuario en cada uno de los componentes.

5.2.2 RF-2. Almacenar código

El código se almacena en nuestro sistema mediante una organización en GitHub, el componente que hace la función de repositorio de código.

5.2.3 RF-3. Descargar código

El componente de GitHub permite el descargarse el código almacenado en el repositorio. Al tener nuestros repositorios públicos cualquier usuario puede realizar la descarga de código.

5.2.4 RF-4. Subir código

GitHub también permite realizar las subidas de código de diferentes modos, ya sea directamente o mediante una petición de integración. únicamente los usuarios autorizados pueden subir código al repositorio.

5.2.5 RF-5. Iniciar ciclo de despliegue

El usuario es el que decide iniciar el ciclo de despliegue realizando un cambio y subiéndolo al repositorio de código. Esto es posible gracias a que Jenkins se encarga de monitorizar que haya cambios en el repositorio iniciando un nuevo ciclo en caso de encontrarlos.

5.2.6 RF-6. Ver el estado del ciclo de despliegue

En Jenkins se puede visualizar a través de su interfaz gráfica de usuario el estado del ciclo de despliegue. Se muestra el estado por cada etapa, indicando el tiempo de ejecución y el estado de las etapas finalizadas.

5.2.7 RF-7. Ejecutar pruebas funcionales de código automáticas

En la fase de test del ciclo de despliegue se ejecutan las pruebas funcionales que el programa lleva incluidas en el código.

5.2.8 RF-8. Visualizar el resultado de las pruebas funcionales

En la interfaz gráfica del Jenkins se puede visualizar la ejecución de todos los pipelines que forman los diferentes ciclos de despliegue. Concretamente se puede ver de cada pipeline la ejecución de todas las fases, entre ellas la de ejecutar las pruebas funcionales. En esta visualización también se muestra el resultado de las pruebas, por lo que se pueden identificar fácilmente los fallos cometidos:

5.2.9 RF-9. Generar artefacto de código

Durante la etapa de empaquetado del ciclo de despliegue Jenkins se encarga de generar el artefacto listo para ejecutarse.

5.2.10 RF-10. Almacenar artefactos de código versionado

Una vez que Jenkins genera un artefacto de código, lo versiona mediante el numero de ejecución del pipeline y lo sube al repositorio de artefactos, que en nuestro caso hemos seleccionado Artifactory.

5.2.11 RF-11. Desplegar artefacto de código automáticamente

La última etapa del ciclo de despliegue es desplegar el artefacto de código generado en un servidor remoto.

5.2.12 RF-12. Dar de alta nuevos usuarios

El súper usuario puede dar de alta nuevos usuarios utilizando el usuario por defecto que cada componente proporciona para llevar a cabo la instalación y configuración inicial del mismo.

5.2.13 RF-13. Acceso a los programas desplegados

El servidor que hemos aprovisionado con Google Cloud tiene el puerto 8080 abierto a la red pública, por lo que si desplegamos la aplicación en ese puerto nuestro programa será accesible por los consumidores de la aplicación.

5.2.14 RNF-1. Cambios nominales en el código

GitHub se encarga de identificar el usuario que realiza cada cambio. De esta manera podemos asociar cada linea del código a un usuario concreto.

5.2.15 RNF-2. Cambios identificados

GitHub también se encarga de asignar un identificador único a cada subida de código. Lo hace mediante un hash, y lo que hace es asociar ese hash al estado del código del repositorio actual.

5.2.16 RNF-3. Tiempo del ciclo de despliegue inferior a cinco segundos

Se han lanzado 10 ejecuciones satisfactorias del ciclo de despliegue y la media de tiempos resultantes por cada etapa son los siguientes:

- Descarga de código: 1 segundo.
- Test: 15 segundos.
- Empaquetado: 10 segundos.
- Subida del artefacto: 883 milisegundos.
- Despliegue del artefacto: 58 segundos.

La suma de las medias de todas etapas resultaría en 84,884 segundos, lo que equivale a 1 minuto y 25 segundos.

CAPÍTULO 6

CONCLUSIÓN

6.1 Conclusión personal

El sector de las tecnologías de la información es un entorno de cambios constantes. Estos cambios se pueden ver tanto en la evolución de las tecnologías como en los procesos que se llevan a cabo. El TFG que hemos llevado a cabo es un claro ejemplo de ello, puesto que es una innovación que se apoya en la tecnología para mejorar los procesos de la organización. Tecnológicamente se ha creado una plataforma de CI/CD que nos sirve para mejorar el proceso de entrega de software.

Este sistema nos facilita poder seguir una metodología DevOps haciendo que el mismo equipo que realiza un cambio en código pueda realizar el despliegue de manera automática y segura sin la necesidad de que otro equipo intervenga, que es lo que sucedía tradicionalmente cuando el equipo de desarrollo realizaba un cambio y tenía que solicitar al equipo de sistemas su puesta en producción.

Al agilizar la puesta en producción de una aplicación y reducir el número de equipos involucrados al generar un cambio desde que se implementa hasta que se despliega estamos mejorando un punto clave del negocio: el tiempo de lanzamiento al mercado. Este punto es muy importante ya que el objetivo de cualquier mejora tiene que ser aportar valor al negocio, puesto que no tendría sentido realizar una mejora en los procesos que no aporte valor a la organización.

La motivación para seleccionar este tema en este TFG ha sido la de dar visibilidad a un proceso de despliegue de software que en el ámbito académico no es muy conocido, sin embargo las organizaciones basadas en tecnologías de la información la realizan. Por experiencia propia, las siguientes empresas líder utilizan herramientas de integración continua orientadas a un modelo DevOps, aunque en ocasiones sin llegar a realizar un despliegue continuo:

- Google.
- Microsoft.

- Amazon Web Services.
- Netflix.
- Spotify.

Personalmente pienso que la selección de este tema ha sido un poco arriesgada ya que es un tema muy técnico en el cual se utilizan muchas tecnologías. Estas tecnologías se han tenido que instalar y configurar para que lleven a cabo las acciones que queríamos que lleven a cabo, pero a la vez hemos tenido que evitar llegar a un nivel de detalle muy bajo para que el TFG no parezca un manual de usuario.

Finalmente, este TFG me ha servido para demostrar que uniendo los conocimientos adquiridos durante el grado junto a los conocimientos obtenidos por experiencia propia en un entorno laboral se puede obtener un gran resultado. En el trabajo realizado se ha intentado unir los conocimientos técnicos sobre una materia aprendidos en un entorno laboral junto a los conocimientos sobre procesos y buenas prácticas aprendidas en el grado.

6.2 Mejoras y siguientes pasos

Realizar una plataforma de CI/CD no quiere decir que estemos trabajando bajo una metodología DevOps. En este TFG hemos trabajado sobre la integración y el despliegue continuo, pero recordemos que esta metodología también abarca otras prácticas como:

- Infraestructura como código.
- Monitorización y registro.
- Microservicios.

Si se deseara mejorar el trabajo realizado incluyendo otros principios de la metodología DevOps se podría profundizar cada uno de estos principios llevando a cabo herramientas y soluciones que aporten valor a la organización. A continuación veremos una serie de proyectos que se podrían llevar a cabo para impulsar un entorno orientado a esta metodología, basándonos en los principios vistos anteriormente. Estos proyectos se van a llevar a cabo

6.2.1 Infraestructura como código

El término de Infraestructura como Código (IAC) significa que toda la infraestructura de software sobre la cual se sustenta la organización está definida bajo algún modelo de datos basado en texto legible por humanos [28]. De esta manera podemos tener nuestra infraestructura plasmada en ficheros repositados en un SCM.

6.2.2 Aprovisionamiento de infraestructura

Mediante distintas tecnologías se pueden procesar estos ficheros para aprovisionar toda la infraestructura que hay definida en ellos. Esta idea es posible gracias a los proveedores de servicios en la nube y a los entornos virtualizados, ya que nos proporcionan

la capacidad de poder solicitar recursos bajo demanda. Cada proveedor de servicios en la nube o cada servicio de virtualización tienen un formato distinto a la hora de solicitar infraestructura, de manera que la especificación de la infraestructura como código en ficheros es diferente para cada uno de ellos.

6.2.3 Configuración de la infraestructura

Éste termino no solo contempla la provisión de infraestructura, sino que también contempla su configuración. Existe un tipo de tecnología que son los sistemas de gestión de la configuración que nos permiten configurar infraestructura ya aprovisionada. La configuración que se lleva a cabo es a nivel de sistemas y son operaciones realizadas sobre los servidores, como la instalación de dependencias para poder ejecutar la aplicación o la modificación de configuraciones del sistema operativo para optimizar su rendimiento.

Contextualizando estos proyectos en el entorno generado para nuestro TFG, podríamos plasmar la especificación del servidor creado en la plataforma de Google Cloud como IAC y repositarlo en nuestra organización de GitHub. En cuanto a la configuración de este servidor, se podría utilizar un gestor de la configuración para especificar en un fichero la instalación los kits de desarrollo de software necesarios para ejecutar nuestros programas, y la instalación de nuestras aplicaciones como servicios en el servidor. Con estas dos acciones se podría incluso automatizar la provisión y la configuración de nueva infraestructura cada vez que se realiza un cambio, lo cual nos abre la puerta a diferentes métodos de despliegue de las aplicaciones.

6.2.4 Monitorización

La monitorización también es una práctica contemplada por la metodología DevOps. La manera de entender la monitorización desde una perspectiva DevOps es creando sistemas para que el mismo equipo que realiza las aplicaciones sea capaz de poder gestionar y visualizar la monitorización de sus sistemas. Para conseguir este hecho podríamos realizar dos sistemas:

- Sistema de métricas.
- Sistema de logs.

6.2.5 Sistema de métricas

Para el sistema de métricas podríamos crear una plataforma que permita enviar métricas definidas en la aplicación a una base de datos de series temporales, la cual a grandes rasgos es una base de datos optimizada para búsquedas por marcas de tiempo. Esta base de datos se puede complementar con alguna herramienta de visualización de métricas, la cual se alimenta de la información de la base de datos para generar paneles de monitorización de métricas.

Es importante para seguir correctamente una metodología DevOps que la plataforma de métricas en su conjunto permita que la creación y configuración de éstas en la base de datos se controle desde la aplicación sin la necesidad de intervención por parte de otro equipo.

6.2.6 Sistema de logs

Otra práctica de monitorización orientada a DevOps es la simplificación del visualizado de los logs de la aplicación. Tradicionalmente los logs residían en los servidores y eran considerados responsabilidad de los equipos de administración de sistemas, pero con la aparición de la metodología DevOps se ha cambiado esta mentalidad y los equipos que llevan a cabo la implementación de una aplicación son los encargados de todo su ciclo de vida. Para permitir a los equipos ver los logs de una manera mas visual y sin requerir conocimientos de sistemas para acceder a los servidores se puede crear una plataforma de logs.

Esta plataforma de logs exporta estos registros de la aplicación del servidor y los muestra en una web de manera visual. Para llevar a cabo este tipo de plataforma se combinan las tecnologías de un exportador de los logs de la aplicación y una base de datos documental optimizada para almacenar documentos de texto. De esta manera, vamos almacenando los registros generados por la aplicación en una base de datos a la cual se le puede añadir una herramienta de visualización para poder realizar consultas sobre la base de datos y obtener los datos de un modo mas visual.

Para implantar estas ideas en el contexto de nuestro TFG, crearíamos una plataforma de métricas con la cual por ejemplo podríamos añadir a nuestras aplicaciones una métrica para registrar cada visita a la ruta expuesta. De este modo el equipo encargado de implementar la aplicación tiene la capacidad de visualizar en un panel cuantas llamadas en el tiempo reciben. También con la creación de una plataforma de logs podríamos ver en una web de manera visual los logs de la aplicación para poder auditar problemas que puedan aparecer.

6.2.7 Microservicios

El principio de los microservicios que impulsa la metodología DevOps es una concepción sobre la arquitectura de software que apoya el diseño de aplicaciones pequeñas en contra de aplicaciones grandes y monolíticas. Por este motivo no hay ningún sistema que podamos llevar a cabo para ayudar a una organización a seguir este principio.

Sin embargo existen algunas tecnologías que van ligadas a los microservicios que nos ayudan a controlar un entorno con aplicaciones de este tipo, como por ejemplo los contenedores o el descubrimiento de servicios.

6.2.8 Contenedores

“Un contenedor es una unidad de software estándar que empaqueta el código y todas sus dependencias para que la aplicación se ejecute de manera rápida y confiable de un entorno informático a otro"[29]. Mediante esta tecnología se facilita el trabajo bajo un entorno asentado en los microservicios facilitando la instalación y la portabilidad de las aplicaciones, ya que este tipo de entornos suele tener un gran volumen de servicios.

6.2.9 Descubrimiento de servicios

El descubrimiento de servicios es un aplicativo que nos sirve para identificar los servicios de nuestro entorno. De esta manera podemos saber en todo momento en que

servidor tenemos instalada cada aplicación, lo cual nos permite tener un mejor control de los recursos consumidos por todo nuestro entorno de servicios.

Para utilizar contenedores en nuestro contexto del TFG tendríamos que crear todo el flujo de CI/CD para aplicaciones en contenedores, lo que incluye las etapas de generar una versión portable de una aplicación basada en contenedores ya compilada, el almacenaje de estas versiones portables, y finalmente su instalación. Todas las acciones para llevar a cabo estas etapas tienen que quedar reflejadas en las tuberías del Jenkins de nuestro proyecto y son distintas a las de la misma aplicación sin la utilización de contenedores. La implantación de un descubrimiento de servicios no tiene mucho sentido en nuestro TFG ya que únicamente nos mostraría dos servicios instalados en el mismo servidor, pero de querer llevarlo a cabo sería sencillo ya que únicamente hay que seleccionar la tecnología que queremos que realice esta función y instalarla utilizando la documentación de dicha tecnología.

BIBLIOGRAFÍA

[1] C. Schaeffer, "Agile vs waterfall," last checked 11/09/2019. [Online]. Available: http://www.crmsearch.com/agile-versus-waterfall-crm.php 1.1

[2] "Amazon web services, what is devops?" last checked 11/09/2019. [Online]. Available: https://aws.amazon.com/es/devops/what-is-devops/ 1

[3] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for agile software development," *The Agile Manifesto*, February 2001, last checked 11/09/2019. [Online]. Available: https://agilemanifesto.org/ 1

[4] I. Kornilova, "Devops methodology," 2017, last checked 11/09/2019. [Online]. Available: https://medium.com/@neonrocket/devops-is-a-culture-not-a-role-be1bed149b0 1.2

[5] I. S. 12208-2008, "Iso/iec 12207:2008," last checked 11/09/2019. [Online]. Available: https://klevas.mif.vu.lt/~adamonis/iso/ISO_IEC_12207_2008(E)-Character_PDF_document.pdf 1, 2.2, 2.4, 3.1.2

[6] I. I. of Business Analysis, "Business analysis body of knowledge (babok)." 2.2, 2.3

[7] I. RedHat, "¿qué son los microservicios?" last checked 11/09/2019. [Online]. Available: https://www.redhat.com/es/topics/microservices 3.3.2

[8] Microsoft, "¿que es una api?" last checked 11/09/2019. [Online]. Available: https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces 3.3.5

[9] Pivotal, "Spring boot framework," last checked 11/09/2019. [Online]. Available: https://spring.io/projects/spring-boot 3.4.1

[10] Microsoft, ".net framework," last checked 11/09/2019. [Online]. Available: https://dotnet.microsoft.com/ 3.4.1

[11] "php built in web server doc," last checked 11/09/2019. [Online]. Available: http://docs.php.net/manual/da/features.commandline.webserver.php 3.4.1

[12] "Python simple httpserver doc," last checked 11/09/2019. [Online]. Available: https://docs.python.org/2/library/simplehttpserver.html 3.4.1

[13] “Languages popularity,” last checked 11/09/2019. [Online]. Available: https://pypl.github.io/PYPL.html 3.4.1

[14] “Languages popularity chart,” last checked 11/09/2019. [Online]. Available: https://www.tiobe.com/tiobe-index/ 3.4.1, 3.2

[15] git scm, “Centralized vs distributed version control (img),” last checked 11/09/2019. [Online]. Available: https://git-scm.com/book/es-ni/v1/Iniciando-Acerca-del-Control-de-Versiones 3.3, 3.4

[16] G. L. Atlassian, “Centralized vs distributed version control,” February 2012, last checked 11/09/2019. [Online]. Available: https://www.atlassian.com/blog/software-teams/version-control-centralized-dvcs 3.4.2

[17] G. Charts, “vcs trends,” last checked 11/09/2019. [Online]. Available: https://trends.google.com/trends/explore?q=git,plasticscm,mercurial 3.4.2

[18] GitHub, “Top projects hosted in github,” last checked 11/09/2019. [Online]. Available: https://octoverse.github.com/projects#repositories 3.4.2

[19] A. W. Services, “Aws codebuild,” last checked 11/09/2019. [Online]. Available: https://aws.amazon.com/es/codebuild/features/ 3.4.3

[20] “Artifactory vs nexus,” last checked 11/09/2019. [Online]. Available: https://www.praqma.com/stories/artifactory-nexus-proget/ 3.4.4

[21] O. S. Michael Larabel, “Windows server 2019 performance benchmarked against linux on an intel xeon server,” last checked 11/09/2019. [Online]. Available: https://www.phoronix.com/scan.php?page=article&item=xeon-windows-2019&num=1 3.4.5

[22] Oracle, “Que es un container?” last checked 11/09/2019. [Online]. Available: https://docs.oracle.com/javase/tutorial/java/annotations/ 4.1

[23] “Maven introduction,” last checked 11/09/2019. [Online]. Available: https://maven.apache.org/ 4.1

[24] “Github organization,” last checked 11/09/2019. [Online]. Available: https://help.github.com/en/articles/about-organizations 4.2

[25] “Jenkins shared libraries,” last checked 11/09/2019. [Online]. Available: https://jenkins.io/doc/book/pipeline/shared-libraries/ 4.3

[26] “Artifactory home page,” last checked 11/09/2019. [Online]. Available: https://jfrog.com/artifactory/ 4.4

[27] “Man scp,” last checked 11/09/2019. [Online]. Available: https://linux.die.net/man/1/scp 4.5

[28] Azure, “Infrastructure as code,” last checked 11/09/2019. [Online]. Available: https://docs.microsoft.com/en-us/azure/devops/learn/what-is-infrastructure-as-code 6.2.1

[29] Docker, “Que es un container?” last checked 11/09/2019. [Online]. Available: https://www.docker.com/resources/what-container 6.2.8