Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Desarrollo de una aplicación web y despliegue automatizado en la nube usando un clúster de contenedores

Estudante: Nicolás Calvo Gómez
Dirección: Roberto Rey Expósito

A Coruña, xuño de 2021.

A mi familia, por apoyarme durante el camino

Agradecimientos

A mi familia y a mis amigos, por toda la ayuda que me han dado durante esta etapa, y a mi director Roberto, por orientarme y dirigirme para poder hacer esto posible.

Resumen

El proyecto realizado en este Trabajo de Fin de Grado (TFG) se puede dividir en dos partes principales. Por un lado, el diseño e implementación de la aplicación Web "Opinews". Esta aplicación se puede considerar una fusión entre periódico y foro, permitiendo a los usuarios de la misma leer las noticias actuales y opinar e interactuar entre ellos gracias a un sistema de comentarios. Por otra parte, se realiza el despliegue de la aplicación en un clúster Kubernetes alojado en la nube pública, Amazon Elastic Kubernetes Service (EKS) en este caso, a través del uso de herramientas que siguen el paradigma Infrastructure as Code (IaC). Estas herramientas nos permiten aprovisionar todos los recursos de infraestructura necesarios para el despliegue de la aplicación Web mediante código fuente de una manera automatizada y simple. Además, se pondrá especial enfásis en el fácil mantenimiento y escalado de la aplicación.

Abstract

The work done in this Bachelor of Science (BSc) Thesis can be divided into two main parts. On the one hand, the design and implementation of the Web application "Opinews". This application can be considered a fusion between a newspaper and a forum, allowing its users to read current news, give their opinions and interact with each other thanks to a comment system. On the other hand, the application is deployed in a Kubernetes cluster hosted in the public cloud, Amazon Elastic Kubernetes Service (EKS) in this case, by using tools that follow the Infrastructure as Code (IaC) paradigm. These tools allow us to provision all the infrastructure resources that are needed for the deployment of the Web application through source code in an automated and simple way. Furthermore, special emphasis will be placed on the easy maintenance and scaling of the application.

Palabras clave:

- Spring Boot
- Infraestructura como código
- Kubernetes
- Terraform
- Amazon Web Services
- Elastic Kubernetes Service

Keywords:

- Spring Boot
- Insfrastructure as Code
- Kubernetes
- Terraform
- Amazon Web Services
- Elastic Kubernetes Service

Índice general

Índice de figuras

Índice de tablas

Listings

Capítulo 1

Introducción

Este primer capítulo se enfoca en poner en contexto este TFG, explicando los motivos principales por los que se decide realizar este proyecto en particular y los objetivos que se proponen alcanzar durante su desarrollo, además de describir la estructura en capítulos de la presente memoria.

1.1 Motivación

La principal motivación para realizar este proyecto en particular surge del gran interés por parte del propio estudiante en los entornos de desarrollo DevOps y las herramientas asociadas a los mismos tras una primera toma de contacto con tecnologías similares durante la carrera. En este trabajo se intenta poner en práctica parte lo aprendido en un escenario de uso más realista, realizando el despliegue de una aplicación Web en una infraestructura computacional alojada en la nube pública utilizando el paradigma de Infrastructure as Code (IaC) [1] con el objetivo de automatizar al máximo posible dicho proceso de despliegue. Además, se hará uso de nuevas y diferentes herramientas IaC no conocidas hasta el momento como son Kubernetes, Terraform y Amazon EKS.

Por otro lado, también se profundiza en el diseño y desarrollo de aplicaciones Web al implementar "Opinews", una fusión entre periódico y foro. Para ello se hará uso del framework Spring junto con Thymeleaf para realizar la implementación utilizando el modelo Modelo-Vista-Controlador (MVC). Esta aplicación Web presenta la complejidad suficiente para que su despliegue en la nube mediante herramientas IaC deba tener en cuenta tanto sus funcionalidades como sus necesidades con el objetivo de adaptar dicho despliegue a las mismas de la manera más óptima posible (p.e. sistema de usuarios, persistencia en base de datos).

1.2 Objetivos

Los objetivos que se proponen en este TFG pueden dividirse en dos grupos:

- Por una parte, se propone el diseño e implementación de la aplicación Web "Opinews", usando herramientas y frameworks adecuados e intentando incorporar ciertas funcionalidades para dotarla de una complejidad representativa de una aplicación real con el objetivo de que su despliegue en la nube deba considerar los aspectos más típicos en estos escenarios.
- Por otra parte, el despliegue de dicha aplicación en la nube pública usando el servicio Amazon EKS, tratando de proporcionar la máxima escalabilidad y disponibilidad posible a la aplicación, además de automatizar al máximo posible todo el procedimiento llevado a cabo. El despliegue completo de la aplicación Web en la nube tiene como meta poder realizarlo con un número mínimo de comandos gracias al uso de herramientas que siguen el paradigma IaC.

1.3 Estructura de la memoria

La estructura en capítulos que presenta este documento es la siguiente:

1. **Introducción:** Pone en contexto el proyecto. Explica las principales motivaciones a la hora de desarrollar este TFG, los objetivos a alcanzar y describe la estructura que sigue la memoria.
2. **Tecnologías y herramientas:** Describe y detalla las principales herramientas y tecnologías usadas a lo largo de este TFG.
3. **Material y metodología:** Lista los materiales necesarios y explica la metodología de trabajo seguida en el desarrollo del proyecto.
4. **Diseño y desarrollo de la aplicación Web:** Este capítulo aborda la primera de las dos partes principales del trabajo, donde se presenta y describe el análisis, diseño e implementación de la aplicación Web "Opinews".
5. **Despliegue de la aplicación Web:** La segunda parte fundamental del proyecto se presenta en este capítulo donde se explican en detalle todos los pasos realizados a la hora de aprovisionar la infraestructura necesaria en la nube en la que se desplegará la aplicación Web, además del diseño de la misma.
6. **Conclusiones**: Finalmente, se exponen las principales conclusiones alcanzadas tras finalizar el trabajo, además de comentar posibles mejoras para la aplicación.

Capítulo 2

Tecnologías y herramientas

En este segundo capítulo se detallan las principales tecnologías y herramientas que fueron usadas a lo largo del proyecto, agrupándolas en tres categorías: las utilizadas en el desarrollo Web (Sección 2.1), las utilizadas en la parte de despliegue en la nube (Sección 2.2) y las que son comunes a ambas (Sección 2.3).

2.1 Desarrollo Web

En esta sección se detallan las tecnologías y herramientas que se han usado durante el desarrollo e implementación de la aplicación Web "Opinews".

2.1.1 Java

Para programar la aplicación Web se ha decidido usar Java, el cúal es uno de los lenguajes de programación más usados en la actualidad, sobre todo en el área de desarrollo Web. Su principal ventaja frente a otros lenguajes es su alta portabilidad, que permite que la aplicación se ejecute sin problemas en cualquier implementación de la Máquina Virtual de Java (Java Virtual Machine, JVM) sin importar la arquitectura hardware de la computadora subyacente. Otras características importantes de Java que cabe mencionar son su orientación a objetos y su recolector de basura, que se encarga de eliminar los objetos automáticamente cuando ya no se están referenciando desde el código fuente.

La elección de este lenguaje de programación se debe, principalmente, a la elección de Spring como framework para facilitar la creación de la aplicación, aunque también a su alta popularidad en entornos de desarrollo Web, lo cúal nos beneficiará a la hora de encontrar documentación en línea y en la resolución de problemas, y por último por la experiencia previa por parte del alumno en dicho lenguaje.

2.1.2 Spring

Spring es un framework de código abierto para el desarrollo de aplicaciones para la plataforma Java, entre otras. Dispone de una gran cantidad módulos y plugins diferentes, los cuales permiten implementar de manera sencilla ciertos aspectos y funcionalidades en un proyecto Java. En nuestro caso, hemos usado los siguientes módulos:

- **Spring Web:** Este módulo nos permitirá implementar la aplicación Web utilizando una metodología Modelo-Vista-Controlador (MVC) y desplegarla localmente en un servidor Tomcat [2].
- **Spring Data JPA:** Este módulo nos permitirá crear entidades y sentencias SQL para relacionar cada objeto del modelo con su tabla en la base de datos [3].
- **Spring Security:** Este módulo nos permitirá implementar un sistema de gestión de usuarios desde nuestra base de datos de manera fácil y simple [4].
- **Spring Boot DevTools:** Este módulo no realiza cambios en la aplicación, pero sirve para desplegarla automaticamente en nuestro servidor local Tomcat tras modificarla.

Además de todo lo anterior, también se ha decidido usar **Spring Tools para Eclipse** como herramienta de programación durante el desarrollo de la aplicación Web.

2.1.3 Maven

Maven [5] es una herramienta software para la gestión y construcción de proyectos y aplicaciones Java, que usaremos en este TFG para manejar las dependencias de los distintos plugins usados (como los propios de Spring) de una manera simple y rápida a través de un fichero XML (pom.xml). Además de eso, también nos permitirá obtener el fichero ejecutable de la aplicación (".jar"), que posteriormente desplegaremos en un clúster de contenedores.

2.1.4 Thymeleaf

Thymeleaf [6] es una biblioteca Java que implementa un motor de plantillas para XML/XHTML /HTML5 que puede ser utilizado tanto en modo Web como en otros entornos. Se acopla perfectamente para trabajar en la capa vista del modelo MVC de las aplicaciones Web. En este TFG se utiliza para realizar ciertas operaciones muy útiles en el fichero HTML, como por ejemplo condicionales, referencias, etc.

2.1.5 HMTL, CSS, JavaScript y JQuery

Primeramente, HTML es considerado el lenguaje Web más popular del mundo, usado para diseñar una estructura básica y un código (denominado código HTML) para la definición de

contenido de una página Web como texto, imágenes, vídeos y juegos, entre otros elementos. Junto a HTML se usa normalmente CSS, un lenguaje de hojas estilos usado para controlar como renderizar y mostrar el código HTML, que de por si solo no resulta especialmente vistoso, permitiendo definir el estilo que tendrán los documentos HTML.

Por otro lado, JavaScript es un lenguaje de scripting (secuencias de comandos) para páginas Web que junto con JQuery, el cual es una biblioteca del anterior, nos permitirán hacer las páginas más cambiantes y dinámicas.

Por último, con el objetivo de hacer el diseño y manejo de estas tecnologías más sencillo, también haremos uso de BootStrap5 [7], el cual es un framework de código abierto que proporciona bibliotecas de CSS y JavaScript que nos permitirá personalizar las vistas de la aplicación más fácilmente al incorporar ciertos elementos ya implementados.

La elección de esta combinación de tecnologías se debe fundamentalmente a su amplia utilización en entornos Web, llegando a ser básicamente un estándar en el diseño del frontend de las aplicaciones Web.

2.1.6 MySQL

MySQL [8] es un sistema de gestión de bases de datos relacional, considerada quizás como la base de datos de código abierto más popular del mundo junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo Web.

La elección de dicha base de datos frente a otras alternativas se debe, principalmente, a la experiencia previa por parte del alumno al haberla utilizado a lo largo de la carrera.

2.2 Despliegue

En esta sección se detallan las tecnologías y herramientas que se han utilizado para automatizar el aprovisionamiento y despliegue de la infraestructura necesaria que alojará la aplicación Web en un entorno de nube pública.

2.2.1 Infrastructure as Code (IaC)

IaC es un paradigma TI mediante el cúal se utiliza código no solo definir para el software, sino también la infraestructura necesaria para su despliegue y ejecución. Es decir, IaC permite definir y aprovisionar recursos de infraestructura a través de código fuente fácilmente legible, en vez de los típicos ficheros o herramientas de configuración, para así poder realizar un despliegue más automatizado y menos propenso a errores.

2.2.2 Kubernetes

Kubernetes (referido en inglés comúnmente como "K8s") es un sistema de código abierto originalmente diseñado por Google para la automatización del despliegue, ajuste de escalado y manejo de aplicaciones que se ejecutan en clusters de contenedores [9]. K8s facilita la automatización y la configuración declarativa de las aplicaciones, y cuenta con un ecosistema grande y en rápido crecimiento.

A diferencia de lo aprendido al trabajar con contenedores Docker durante la carrera, con Kubernetes tendremos que aprender a definir diferentes tipos de estructuras que serán usadas dentro del clúster para conseguir un correcto funcionamiento. Algunas de las más importantes para este TFG son:

- **Deployments:** Pueden ser definidos como controladores cuya función es obtener y mantener un "estado deseado" del despliegue [10].
- **Pods:** Son un grupo de uno o más contenedores donde se ejecutará la aplicación una vez desplegada. Representan las unidades de computación desplegables más pequeñas que se pueden crear y gestionar en Kubernetes [11].
- **ConfigMaps:** Son utilizados para almacenar datos no confidenciales en el formato clave-valor. Pueden ser usados en los Deployments como variables de entorno, argumentos de la línea de comandos o como ficheros de configuración en un Volumen. Permiten desacoplar la configuración de un entorno específico de una imagen de contenedor, haciendo que las aplicaciones sean fácilmente portables[12].
- **Volumes:** Sirven como unidad de almacenamiento en la que podemos guardar ciertos ficheros para proporcionarselos a nuestros Pods [13].
- **Secrets:** Parecidos a los ConfigMaps, pero usados para almacenar y administrar información de carácter confidencial (p.e. contraseñas, llaves ssh) [14].
- **PersistentVolumes:** Volumenes independientes que no se borran al eliminar un Pod o Deployment. Por tanto, permiten dotar a la aplicación de persistencia como su nombre hace intuir [15].
- **Services:** Permiten configurar el acceso a los Pods, definiendo los puertos que se usarán, el tipo de servicio, etc [16].

Un clúster Kubernetes sigue una arquitectura master/worker similar a la mostrada en la Figura 2.1. Los nodos worker son los encargados de ejecutar los contenedores dentro de las estructuras explicadas anteriormente, que son los Pods. El estado de un Pod puede gestionarse

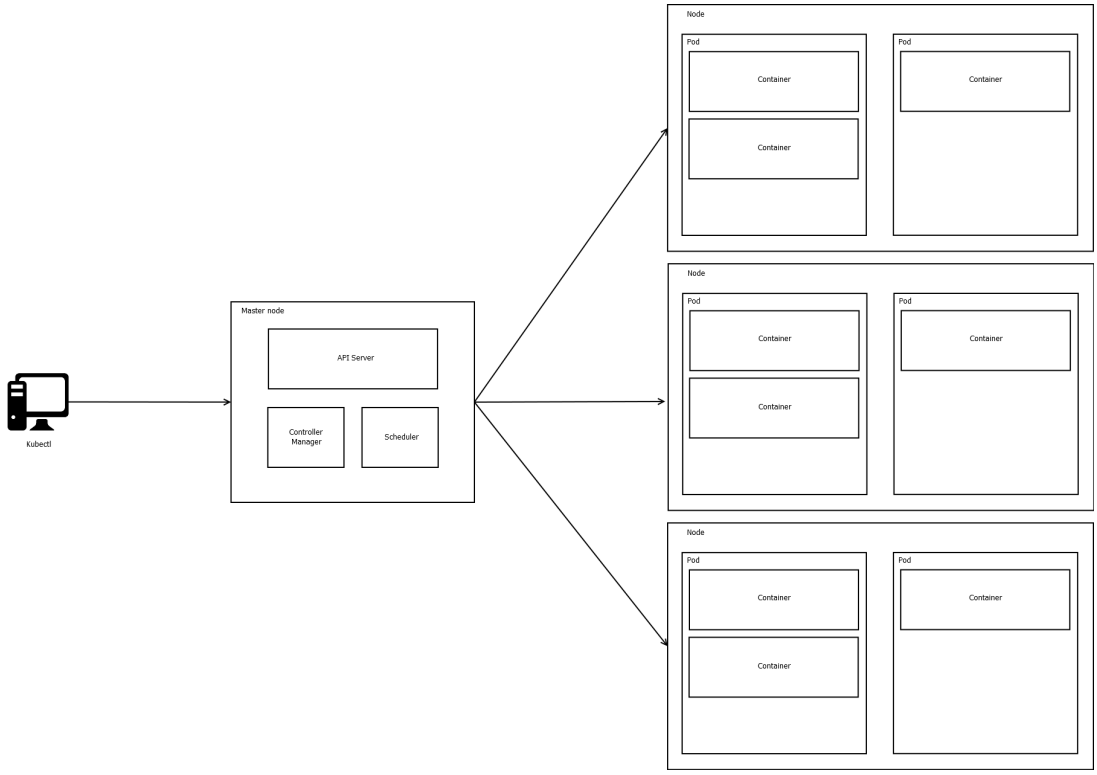

Figura 2.1: Arquitectura general de un clúster Kubernetes

a través de los Deployments, y su acceso se configura a través de los Services. La relación entre los distintos componentes mencionados se puede ver en la Figura 2.2.

La decisión de usar Kubernetes en este proyecto se debe a su enorme popularidad en la actualidad para el despliegue y escalado de aplicaciones en entornos virtuales basados en contenedores, y en la que el alumno tenía especial interés de antemano.

2.2.3 Amazon Web Services

Amazon Web Services (AWS) [17] es una colección de servicios que, en conjunto, conforman una plataforma de computación en la nube ofrecida a través de Internet y mediante un modelo de pago por uso. En la actualidad, AWS es el proveedor público de servicios de computación en la nube más popular y utilizado del mundo.

En este TFG, los servicios de AWS no serán desplegados de forma manual a través de su interfaz Web, sino que se gestionarán a través de la herramienta IaC Terraform para lograr así un despliegue automatizado de la aplicación Web desarrollada.

Los principales servicios de AWS usados en este proyecto son:

- **Elastic Compute Cloud (EC2):** Este servicio se puede considerar el corazón de AWS, pues permite crear máquinas virtuales bajo demanda para la ejecución de todas las

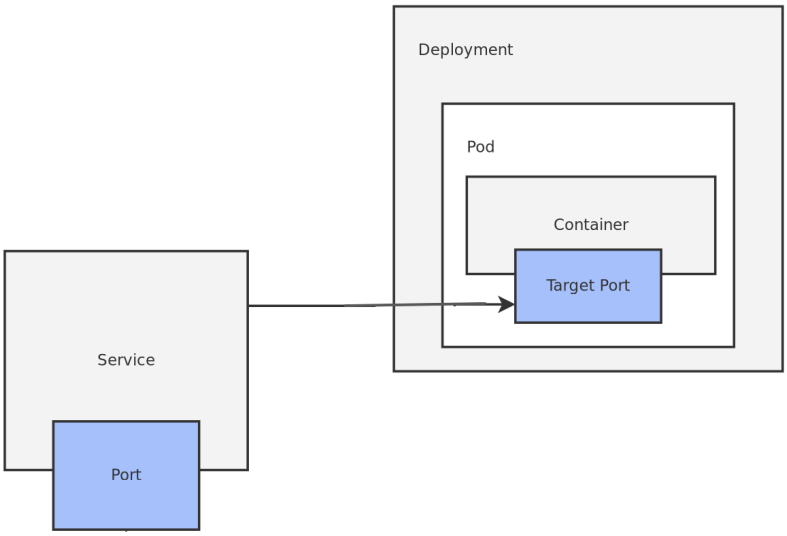

Figura 2.2: Relación entre Service-Deployment-Pod-Container

operaciones necesarias de nuestro clúster Kubernetes y la aplicación Web. EC2 permite utilizar una gran variedad de sistemas operativos, gestionar permisos de acceso a la red y ejecutar tantas instancias como se desee ya que se trata de un servicio elástico y escalable en el que es posible crear, lanzar y finalizar tantas instancias virtuales como se necesite [18].

- **Elastic Kubernetes Service (EKS):** Este servicio se puede considerar el más importante para este TFG. Permite desplegar de manera sencilla un clúster Kubernetes seguro y de alta disponibilidad en AWS en el que poder publicar nuestra aplicación Web, gracias a la instalación y configuración automática de Kubernetes en una o varias instancias EC2 [19].
- **Virtual Private Cloud (VPC):** Este servicio nos proveerá de una red privada en la cual estará desplegado nuestro clúster y sus instancias EC2 [20].
- **LoadBalancer:** Este servicio se utilizará para enrutar el tráfico entrante al destino de manera más eficiente, repartiendo las peticiones entre múltiples instancias [21].

2.2.4 Terraform

Terraform [22] es una herramienta IaC que permite a los usuarios definir, aprovisionar y configurar toda la infraestructura de un centro de datos en un lenguaje declarativo de alto

```
resource "aws_instance" "iac_in_action" {
  ami               = var.ami_id
  instance_type     = var.instance_type
  availability_zone = var.availability_zone

  // dynamically retrieve SSH Key Name
  key_name = aws_key_pair.iac_in_action.key_name

  // dynamically set Security Group ID (firewall)
  vpc_security_group_ids = [aws_security_group.iac_in_action.id]

  tags = {
    Name = "Terraform-managed EC2 Instance for IaC in Action"
  }
}
```

Listing 2.1: Ejemplo de uso de Terraform

nivel, generando un plan de ejecución que permite desplegarla de manera fácil y automatizada en diferentes plataformas de la nube como AWS, Azure, Google Cloud, entre otras.

Dicho plan de ejecución generado por Terraform describe cual es el estado deseado a alcanzar, comparándolo con el estado ya existente (en caso de haberlo) y realizando las modificaciones necesarias para aprovisionar y/o modificar la infraestructura. Esto es posible gracias a todos los módulos y plugins que pueden descargarse, que permiten a Terraform aumentar la cantidad de recursos de infraestructura que puede manejar y adaptarse así a las particularidades cada despliegue.

En el fragmento de código 2.1 se muestra un pequeño ejemplo de la sintaxis usada para la declaración de una instancia EC2 en AWS con Terraform.

2.2.5 Docker

Docker [23] es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. Esta herramienta se usará en este TFG para crear la imagen Docker que empaquetará nuestra aplicación Web junsto con sus dependencias, y se subirá dicha imagen a DockerHub (un repositorio de imágenes público y gratuíto). Dicha imagen será descargada y desplegada posteriormente en el clúster Kubernetes.

2.2.6 Visual Studio Code

Visual Studio Code [24] es un editor de código fuente multiplataforma desarrollado por Microsoft para entornos Windows, Linux y macOS. Incluye ciertas herramientas muy útiles

como soporte para la depuración, control integrado de Git, resaltado de sintaxis, y finalización inteligente de código. Además, permite una gran personalización gracias a una gran cantidad de plugins que permiten personalizar la herramienta o agregarle todavía más características.

2.3 Comunes

En esta sección se agruparán las herramientas usadas tanto en la parte de desarrollo como en la de despliegue.

2.3.1 Git

Git es el sistema de gestión de versiones más usado del mundo. Sus características más importantes son:

- Es gratuito, por lo que no supone costo alguno.
- Presenta una arquitectura distribuida, lo que permite tener un repositorio local en el equipo que puede albergar el historial completo de todos los cambios.
- Permite el uso de etiquetas y un desarrollo en ramas, lo que facilita mucho el desarrollo de las aplicaciones, sobre todo cuando trabajan más de una persona en ellas.

En el proyecto Git será usado junto con el servicio de gestión de repositorios GitHub, el cual es uno de los más populares en la actualiad. En dicho repositorio se irá guardando todo el progreso hecho tanto en la aplicación Web como en los ficheros resultado del despliegue de la misma.

Capítulo 3

Material y metodología

En este capítulo se especifican los materiales hardware y software necesarios para llevar a cabo el trabajo y la metodología que se siguió durante su realización.

3.1 Materiales

Para poder realizar este proyecto son necesarios los siguientes materiales:

- Hardware: Ordenador personal, con cualquier sistema operativo, pues vamos a utilizar un lenguaje multiplataforma. No se requieren recursos hardware específicos, pues la aplicación Web se ejecutará en la nube.
- Software: Java JDK (versión 14 recomendada), Maven, Terraform, Docker, Spring Tools 4, Virtual Studio Code, Git y cuentas creadas en los siguientes servicios: AWS, DockerHub y GitHub.

Además de eso, para la realización de los bocetos se ha utilizado Balsamiq Mockups 3, herramienta de pago que dispone de una versión de prueba, con la cual se pueden crear varias vistas prototipo y relacionarlas entre sí, lo que facilita el desarrollo al tener una guía que seguir y proporciona una mejor comunicación con el cliente, en caso de tenerlo.

Para el diseño del diagrama modelo relacional se ha utilizado Dia, una aplicación de código abierto especializada en la creación de múltiples diagramas.

3.2 Metodología

En este trabajo se ha decidido seguir una metodología de desarrollo iterativa incremental, en la que dividiremos todas las tareas a realizar en bloques temporales denominados iteraciones. En cada una de ellas, se llevará a cabo un procedimiento similar: análisis, diseño,

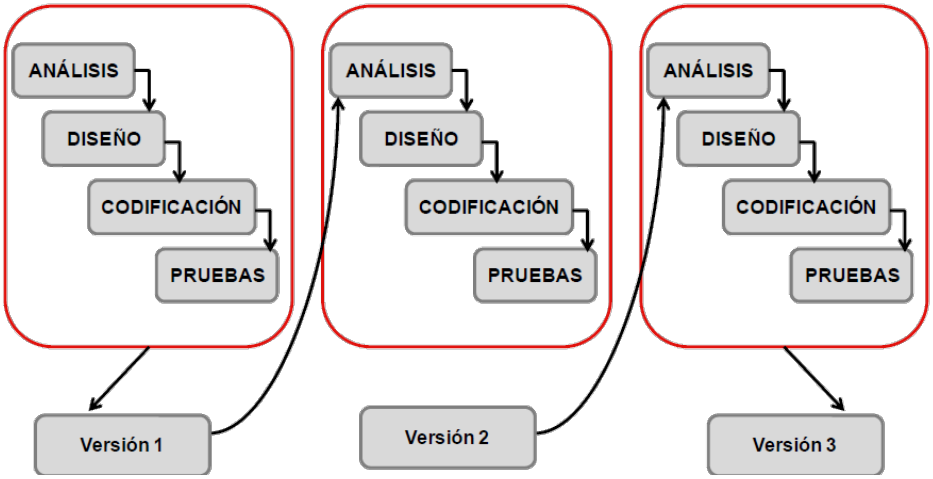

Figura 3.1: Metodología iterativa incremental

codificación y pruebas (véase Figura 3.1). Esta metodología posibilita implementar las funcionalidades de la aplicación desde las más sencillas hasta la más complejas, lo que también permite centrarse más en el manejo de ciertas tecnologías antes de pasar a otras, fragmentando así el proceso de aprendizaje y haciéndolo más sencillo. En el caso de tener un cliente, este estilo de desarrollo nos permitiría ir concertando reuniones en las que se le mostraría al cliente las funcionalidades ya desarrolladas, para así obtener su opinión o "feedback" y poder ir refinando el producto el sucesivas iteraciones.

3.2.1 Planificación

La parte de desarrollo de la aplicación Web se ha dividido en las siguientes iteraciones:

1. Prototipado: En esta primera iteración se definirán los requisitos funcionales y se realizarán los bocetos y diagramas que servirán como guía para el desarrollo de la aplicación Web.

2. Página principal: Esta iteración servirá como una primera toma de contacto con las tecnologías que usaremos en el desarrollo de la aplicación, por lo que el objetivo será comprender su funcionamiento mientras se implementa únicamente la página principal de la misma.

3. Integración con la base de datos: En esta iteración se conectará la aplicación ya existente con la base de datos que será usada en el entorno local.

4. Operaciones CRUD: Tras la conexión a la base de datos realizada en la iteración anterior, se procederá a implementar las operaciones Crear-Leer-Actualizar-Borrar (Create-

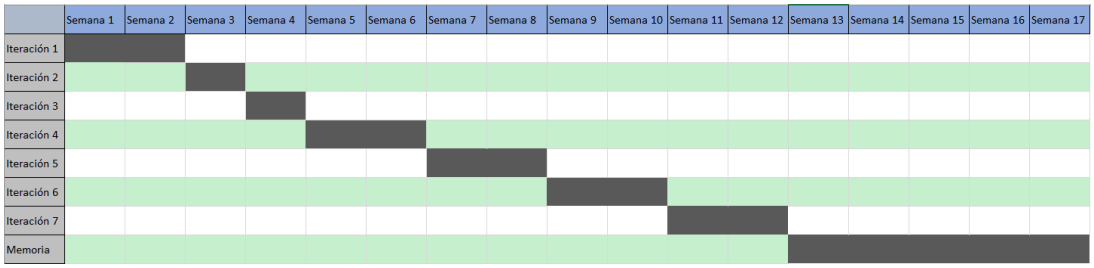

Figura 3.2: Cronograma con la duración estimada de las iteraciones

Read-Update-Delete, CRUD) para los objetos que se manipulan en la aplicación.

5. Sistema de usuarios: Por último, se implementará un sistema de usuarios con diferentes roles que permitan restringir las acciones que puede realizar cada tipo de usuario.

Por otra parte, el despliegue de la aplicación Web en el clúster Kubernetes de contenedores estará dividido en dos iteraciones:

1. Despliegue en local: Se realizará un primer despliegue de la aplicación Web utilizando un clúster de Kubernetes instalado en local, con el objetivo de aprender sobre Kubernetes, su funcionamiento básico y componentes sin que ello suponga costes adicionales.

2. Migración a EKS: Tras comprender el funcionamiento de Kubernetes, se comenzarán a usar las herramientas de Terraform y AWS/EKS para replicar el despliegue en la nube pública utilizando el paradigma IaC.

La duración estimada de todas las iteraciones se refleja en el cronograma mostrado en la Figura 3.2, donde también se incluye el tiempo necesario para la redacción del presente documento.

Capítulo 4

Diseño y desarrollo de la aplicación Web

En este capítulo se describe todo el proceso llevado a cabo tanto para el diseño como para el desarrollo de la aplicación Web "Opinews", que será desplegada posteriormente en un entorno de nube pública. Como se ha mencionado en el capítulo previo (véase Sección 3.2), el desarrollo de la aplicación se ha dividido en varias iteraciones en las que se implementarán una o varias funcionalidades de la misma.

4.1 Primera iteración: Prototipado

Antes de empezar a programar el código de la aplicación Web, primero hay que analizar las funcionalidades principales que queremos que proporcione a sus potenciales usuarios, así como realizar un pequeño boceto con un primer diseño de la vista.

4.1.1 Funcionalidades

En la Tabla 4.1 se listan los requisitos funcionales que se tiene pensado implementar en la aplicación Web.

4.1.2 Bocetos

Además de las funcionalidades deseadas para la aplicación, también es recomendable elaborar una serie de bocetos conceptuales para ayudarnos a tener una mejor idea a la hora de diseñar las vistas con HTML+CSS (y en caso de tener un cliente, poder enseñarselos para obtener su visto bueno, hacer cambios antes de empezar, etc). A continuación, se muestran los bocetos de las vistas a crear en la Figuras 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9 y 4.10.

ID	Descripción
RF1	Registrar usuario
RF14	Iniciar sesión
RF2	Crear noticia
RF3	Ver noticia
RF4	Ver posts
RF5	Listar noticias
RF6	Listar posts
RF7	Filtrar noticias
RF8	Filtrar posts
RF9	Modificar noticia
RF10	Modificar posts
RF11	Eliminar noticia
RF12	Eliminar posts
RF13	Crear comentario
RF14	Modificar comentario
RF14	Eliminar comentario
RF15	Listar comentarios del post

Tabla 4.1: Requisitos funcionales de la aplicación Web

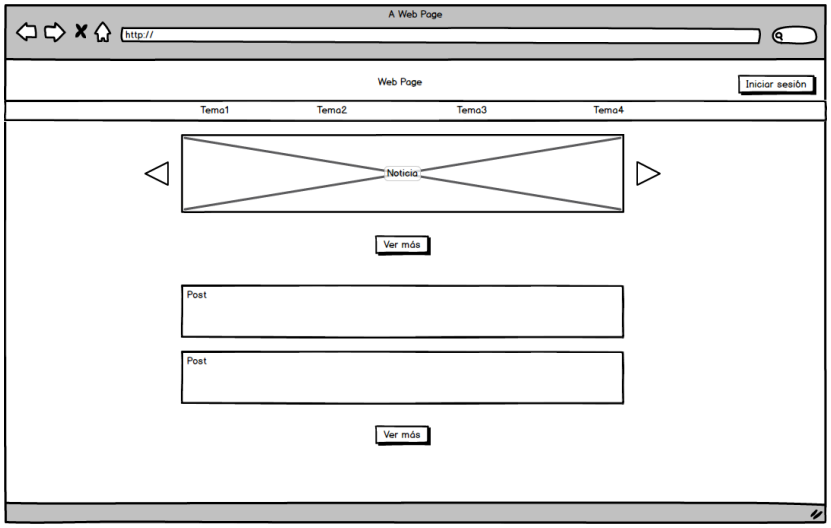

Figura 4.1: Boceto de la página principal de la aplicación

Además de los bocetos, se muestra en la Figura 4.11 el diseño lógico de la base de datos mediante un diagrama del modelo relacional de la misma. Se puede observar que tenemos una tabla para cada elemento diferente que manejamos en la aplicación (Noticias, Posts, Temas, Comentarios y Usuarios) y las diferentes relaciones entre ellos (qué usuario creó cada noticia, a qué tema pertenecen, a qué post pertenece un determinado comentario, etc).

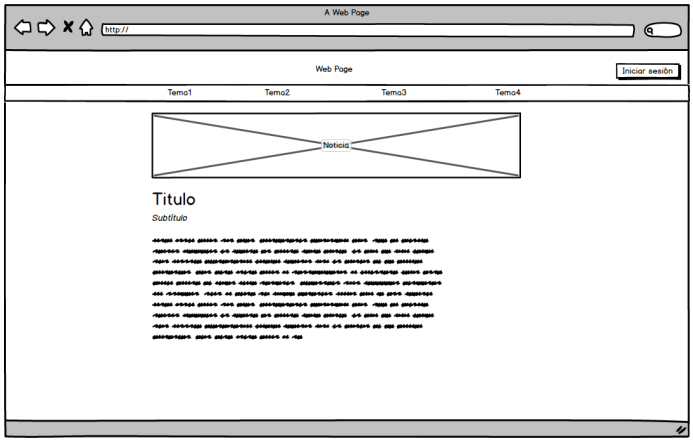

Figura 4.2: Boceto de la página para leer una noticia

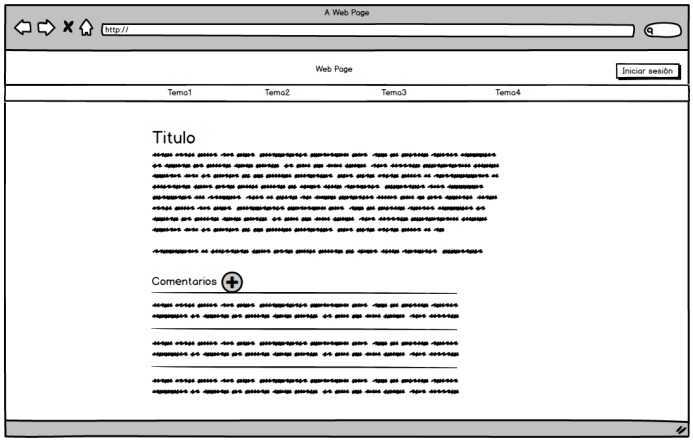

Figura 4.3: Boceto de la página para leer un post

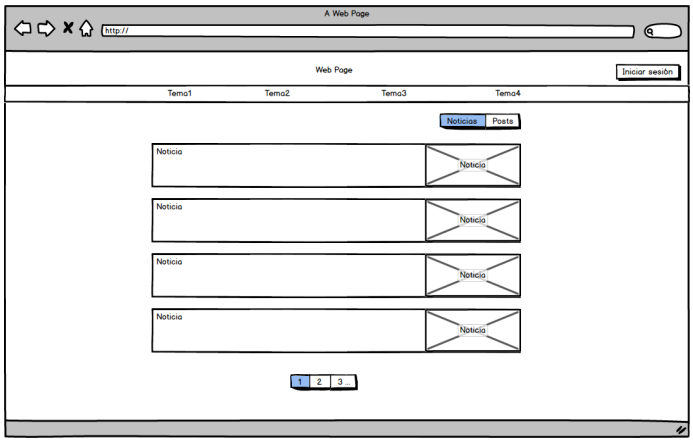

Figura 4.4: Boceto de la página de lista de noticias

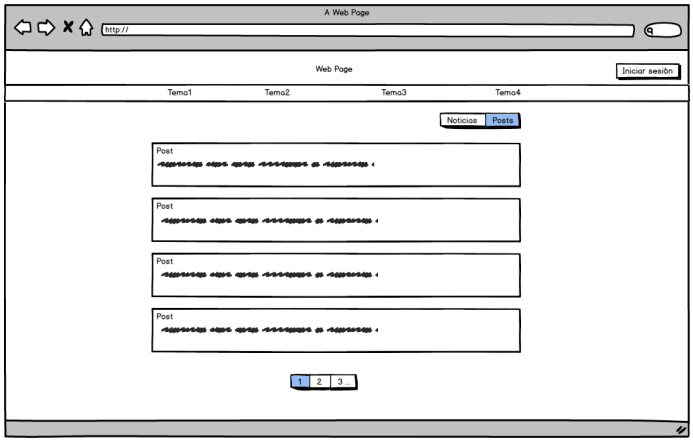

Figura 4.5: Boceto de la página de lista de posts

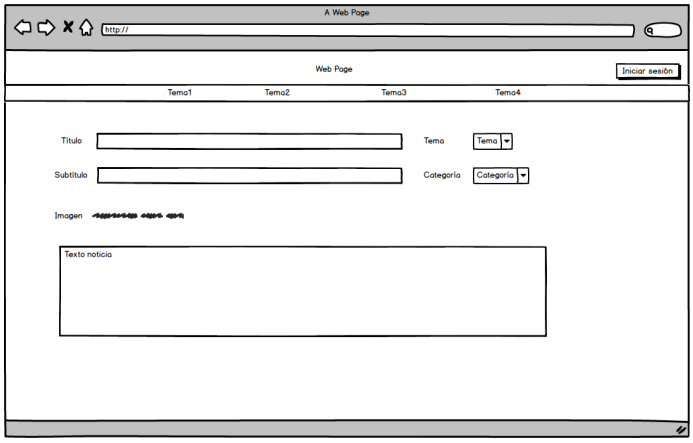

Figura 4.6: Boceto del formulario de creación de noticia

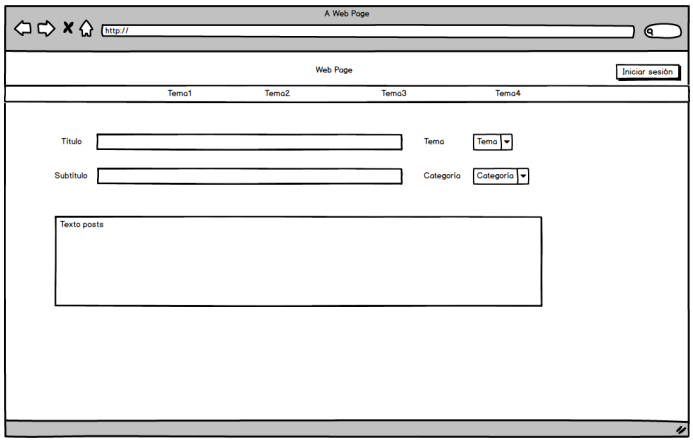

Figura 4.7: Boceto del formulario de creación de posts

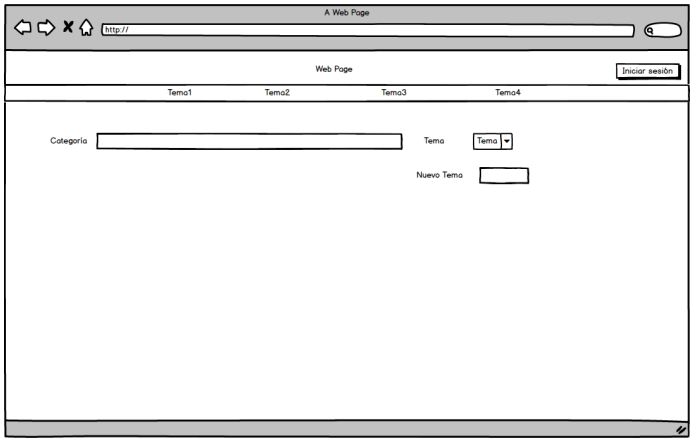

Figura 4.8: Boceto del formulario de creación de temas

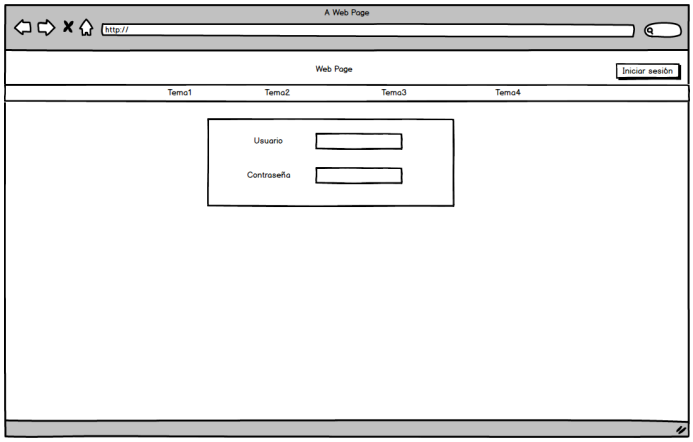

Figura 4.9: Boceto de la página de inicio de sesión

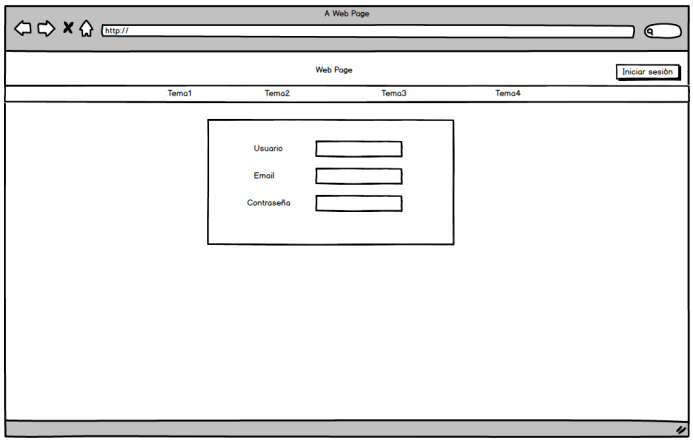

Figura 4.10: Boceto de la página de registro de usuario

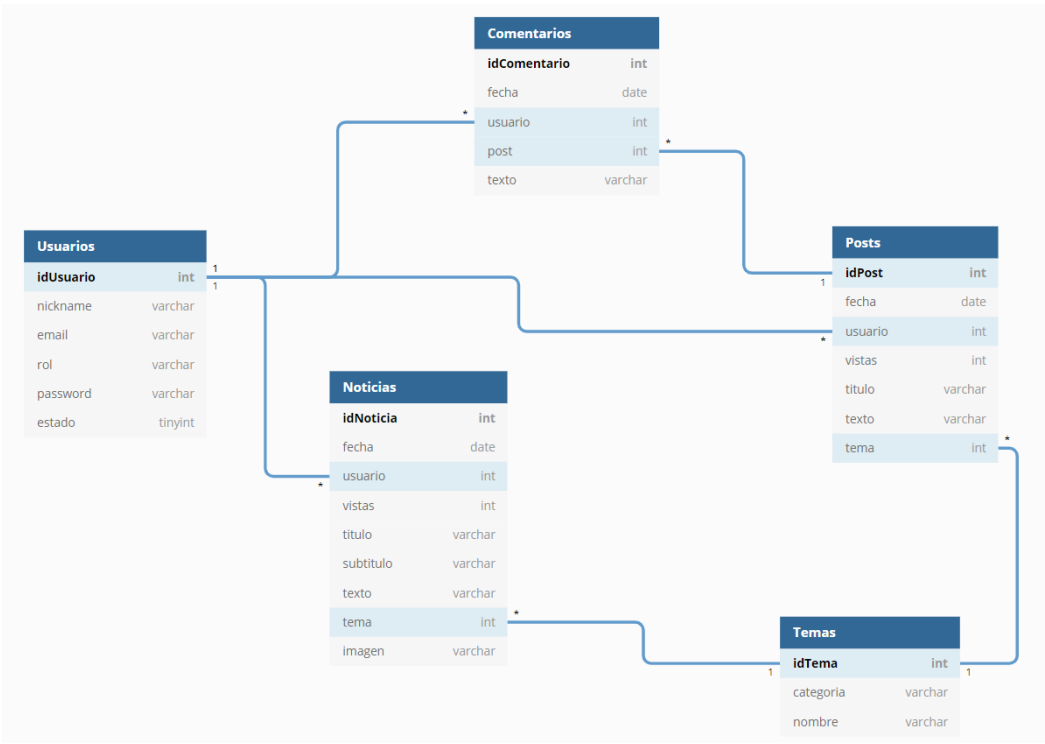

Figura 4.11: Diagrama del modelo relacional de la base de datos

4.2 Segunda iteración: Página principal

4.2.1 Descripción e implementación

Esta segunda iteración, a pesar de que solo se desarrollará la página principal de la aplicación Web, puede ser perfectamente la más compleja de todas, pues es necesario familiarizarse con las principales tecnologías utilizadas durante la fase de desarrollo, como son Spring Web, Spring JPA, Thymeleaf, etc.

Para empezar, tras instalar Spring Tools 4, utilizaremos la página Web de Spring Initializr (véase Figura 4.12), la cual nos permite crear un proyecto Spring con las dependencias necesarias de una manera rápida y simple. En nuestro caso, empezaremos con las dependencias de Spring Data JPA y Spring Security desactivadas hasta que se comprenda mejor el funcionamiento de las restantes herramientas a usar, lo que dará lugar a una etapa de aprendizaje algo más sencilla al reducir la carga inicial.

Una vez que el proyecto está importado y la dependencia desactivada (comentada o sin añadir en el fichero pom.xml de Maven), empezaremos realizando un código HTML muy sencillo con la cadena "Hola mundo", y se intentará que se muestre al usuario. Para ello, se debe editar el fichero HTML y moverlo a la carpeta */src/main/resources/templates* de nuestro pro-

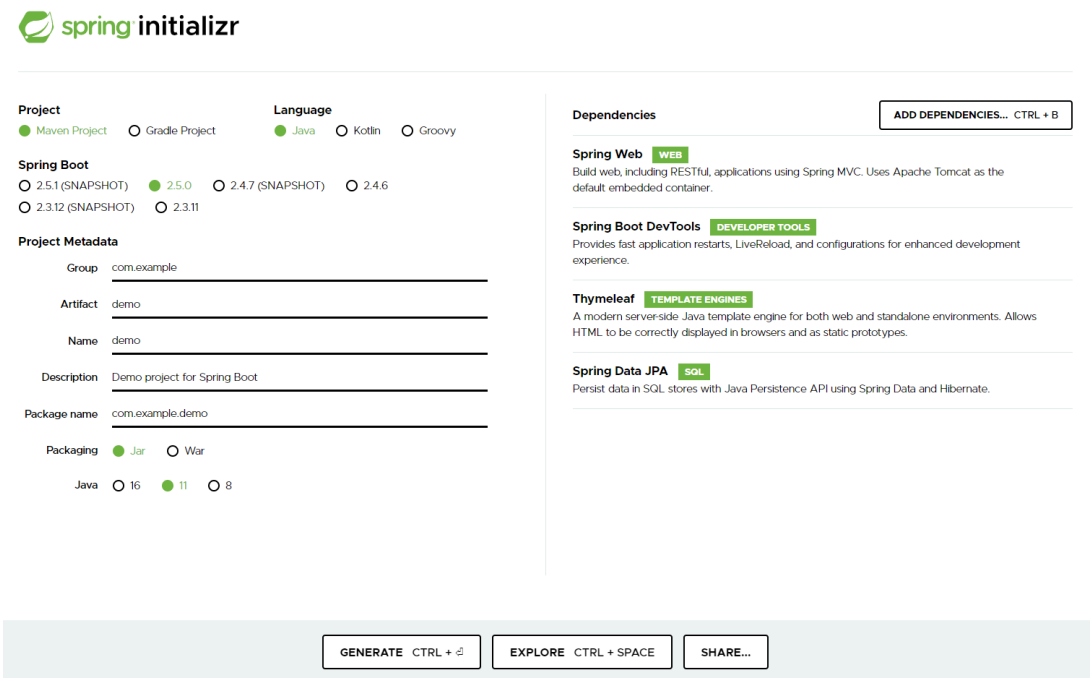

Figura 4.12: Ejemplo de Spring Initializr

yecto. Es en esta ruta donde Thymeleaf buscará las plantillas a utilizar. Seguidamente, se crea un controlador que será el encargado de devolver el fichero HTML al acceder a una URL específica. Para realizar las pruebas, se hará uso del servidor local Tomcat que viene integrado por defecto y el cual se actualiza automáticamente al realizar cambios en el código gracias a la dependencia de Spring Boot DevTools.

Tras haber logrado lo anterior, es posible empezar a personalizar el fichero HTML conforme al boceto realizado anteriormente. Para dicho propósito se hace uso de BootStrap5, una biblioteca de CSS que nos permitirá aplicar diferentes estilos de manera fácil y rápida. Los ficheros se almacenan en la carpeta */src/main/resources/static/bootstrap* y se cargan en el código HTML mediante referencias de Thymeleaf (p.e. *<link th:href="@/bootstrap/css/bootstrap-blog.css" rel="stylesheet">*). También es necesario añadir Thymeleaf al principio del fichero mediante la sentencia *<html xmlns:th="http://www.thymeleaf.org">*, lo que nos permitirá realizar ciertas funciones muy útiles en el código HTML (condicionales, referencias, acceso a datos del modelo, etc).

Una vez realizados todos estos pasos, el resultado final sería el de la Figura 4.13.

4.2.2 Observaciones

Esta iteración ha servido como primer punto de contacto con la mayoría de tecnologías a usar en la fase de desarrollo de la aplicación. Aunque el título especifique como objetivo la

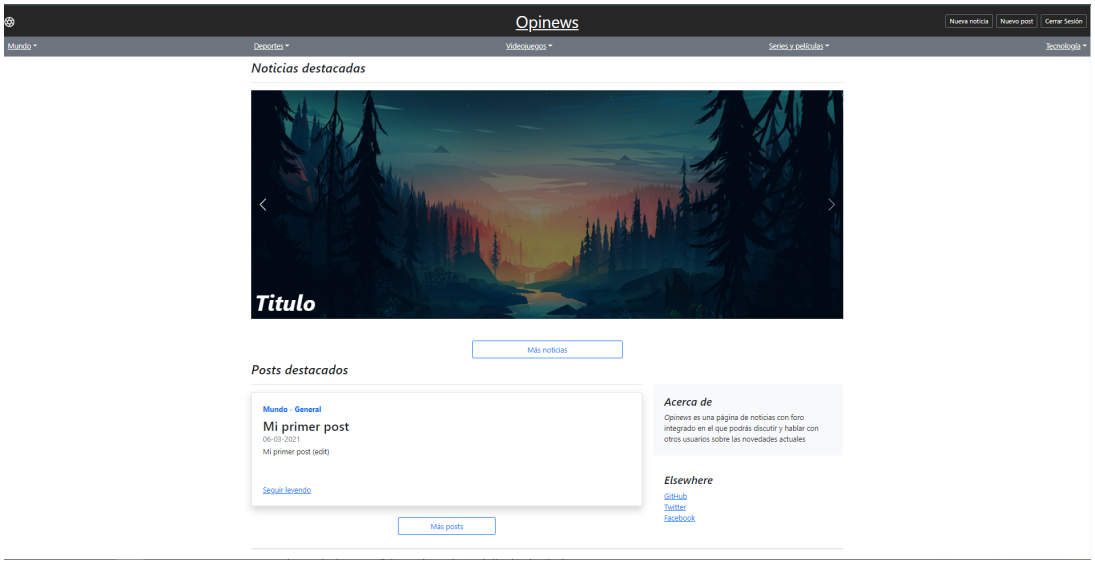

Figura 4.13: Resultado de la página principal

realización de únicamente la página principal, la verdadera finalidad de la iteración es obtener práctica y soltura con Spring, ficheros HTML y Thymeleaf para comprender su funcionamiento, lo cual reflejaremos en la creación de la vista objetivo.

Como resultado, el alumno obtuvo conocimiento y soltura con dichas herramientas, lo que ocasionará que las demás iteraciones se realicen en un intervalo de tiempo menor.

4.3 Tercera iteración: Integración con la base de datos

4.3.1 Descripción e implementación

Una vez tenemos la página principal personalizada, es posible activar la dependencia Spring Data JPA para empezar a recuperar elementos de la base de datos.

Primeramente, hay que crear la base de datos de acuerdo al diagrama mostrado anteriormente (véase Figura 4.11). A continuación, se deben añadir algunos valores válidos a la misma para simular noticias o posts ya creados. Tras esto, es posible activar la dependencia en el fichero pom.xml y configurar la conexión con la base de datos en el archivo properties, donde le indicaremos el driver, URL, usuario, contraseña, estrategia, etc. (véase Figura 4.14). A continuación, se crea una interfaz por cada entidad de la base de datos, las cuales extenderán la clase JpaRepository<*Nombre clase*,*Tipo ID*>. Gracias a ella es posible hacer uso de ciertas funciones ya definidas, así como crear nuestras propias funciones personalizadas (véase un ejemplo en la Figura 4.15).

Cuando la conexión a la base de datos sea funcional, faltaría por configurar las clases

```
# DATASOURCE (MYSQL 8.0)
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/ExampleApp?useSSL=false&serverTimezone=Europe/Madrid&allowPublicKeyRetrieval=true
spring.datasource.username=root
spring.datasource.password=root
#JPA
spring.jpa.generate-ddl=false
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.show-sql=false
# Table names physically
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
```

Figura 4.14: JPA application.properties

```
package com.App.repository;

import java.util.List;

public interface TemasRepository extends JpaRepository<Tema, Integer> {

    @Query("SELECT categoria, min(idTema) as id from Tema group by categoria order by id")
    List<String> findDistinctByCategoria();

    @Query("select t from Tema as t where categoria= :cat ORDER BY CASE WHEN nombre = 'General' THEN 1 ELSE 2 END, nombre")
    List<Tema> findTemaByCategoria(@Param("cat") String categoria);
}
```

Figura 4.15: Interfaz *TemasRepository*

del modelo con las anotaciones correspondientes para que JPA identifique las entidades y sus atributos correctamente (véase Figura 4.16). Tras obtener los datos, el siguiente paso es añadirlos a un objeto Model en el método correspondiente del controlador, para así poder acceder a sus propiedades en el código HTML con Thymeleaf (p.e. *th:text=${noticia.titulo}*).

4.3.2 Observaciones

En esta iteración se ha investigado y probado la herramienta de Spring Data JPA para crear la capa de acceso a la base de datos, aprendiendo además cómo mostrar los elementos en el código HTML utilizando Thymeleaf. Con esta iteración y la anterior, se dispone de todo el conocimiento necesario para empezar a desarrollar los métodos y vistas de las operaciones CRUD para las noticias y los posts.

4.4 Cuarta iteración: Operaciones CRUD

Una vez que se comprende el funcionamiento de las tecnologías que se han decidido utilizar, se pasa a realizar las vistas y métodos para las operaciones CRUD.

Para ello, primero se debe aprender el funcionamiento de los formularios de HTML y Thymeleaf, mediante los cuales se le pasan los datos a insertar o modificar al controlador. Para este propósito, se crean unos inputs de prueba y se imprime el resultado en la consola de Spring Tools para comprobar el correcto funcionamiento.

Una vez que se entiende cómo pasar los datos (incluyendo la imagen de portada de las noticias y el editor para formatear el texto), se procede al diseño de las vistas consultando los

```java
package com.App.model;

import java.util.Date;

@Entity
@Table(name="Posts")
public class Post {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer idPost;
    private Date fecha = new Date();
    @ManyToOne
    @JoinColumn(name="usuario")
    private Usuario usuario;
    private Integer vistas = 0;
    private String titulo;
    private String texto;
    @ManyToOne
    @JoinColumn(name="tema")
    private Tema tema;
    private String imagen;
```

Figura 4.16: Anotaciones JPA en clases del modelo

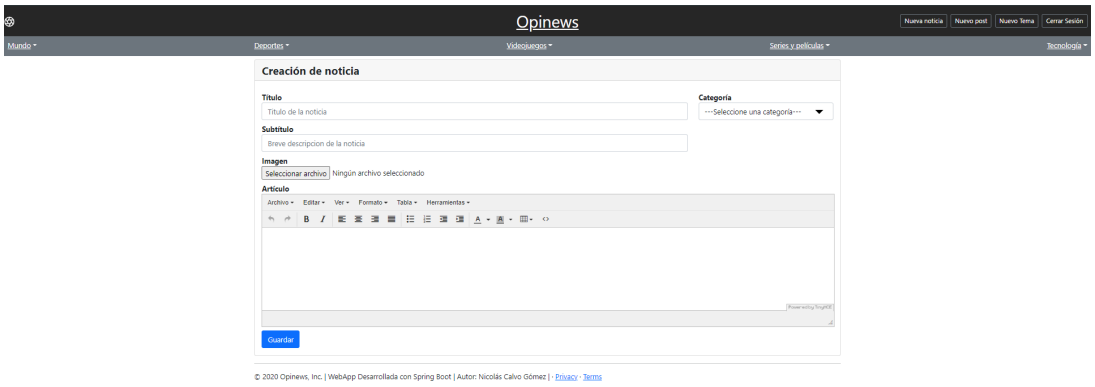

Figura 4.17: Ejemplo de la página de creación de noticias

bocetos mostrados previamente. Al termino de este proceso, se tendrían las páginas necesarias para realizar las operaciones de crear, leer, modificar y eliminar (Figuras 4.17 y 4.18).

Por último, hay que conectar todas las vistas entre sí mediante las referencias correspondientes con Thymeleaf (por ejemplo, acceder a la vista de la noticia desde la página principal), y también comprobar el correcto funcionamiento de todas ellas una vez más.

4.4.1 Observaciones

En esta iteración se utilizaron las tecnologías que habían sido exploradas en las iteraciones anteriores para terminar las vistas que faltaban y conectarlas entre sí mediante referencias

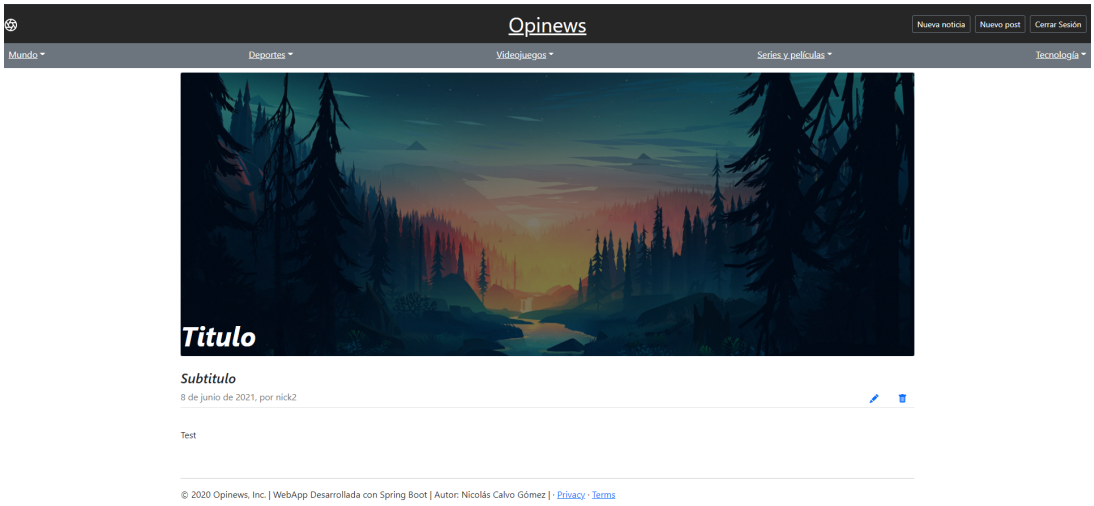

Figura 4.18: Lectura de una noticia, con los iconos para modificar y eliminar

HTML. Al terminar la iteración, el resultado debería ser una aplicación funcional la cual cumpliría casi todos los requisitos funcionales especificados previamente en la Tabla 4.1, excepto los relacionados con los usuarios de la aplicación. Estos requisitos se implementarán en la siguiente y última iteración.

4.5 Quinta iteración: Sistema de usuarios

Una vez disponemos de una página funcional, se procede a implementar el sistema de usuarios. Este sistema nos permitirá tanto crear un usuario nuevo en la aplicación como restringir el acceso a ciertas funcionalidades dependiendo del tipo de usuario o rol que se esté utilizando. Para ello, se hace uso de la dependencia de Spring Security, por lo que debe volverse a activar en el fichero pom.xml del proyecto (o añadirla de nuevo, en caso de haber sido eliminada).

Antes de continuar, es necesario documentarse acerca del funcionamiento de esta dependencia, y decidir cómo se desea implementar el sistema de usuarios. En nuestro caso, se ha decidio permitir que cualquier persona (usuario o no) pueda leer cualquier publicación de la página, aunque en lo referente a la creación de las mismas sí diferenciaremos entre tres roles diferentes. Estos son:

- **Usuario:** Usuario común que solo podrá crear posts y comentarios.
- **Periodista:** Usuario encargado de la creación de noticias y que solo podrá crear publicaciones de este tipo.

Figura 4.19: Página personalizada de inicio de sesión

- **Administrador:** Usuario diseñado para su uso por parte de los administradores de la página. Se le permitirá realizar todas las opciones posibles, desde crear posts, comentarios y noticias a la creación de temas y categorías nuevas.

En primer lugar, es necesario configurar la tabla de la base de datos desde la que se obtendrán los campos (tabla "Usuarios" en este caso). Esta tabla debe tener como mínimo un nombre de usuario, contraseña y un campo de tipo entero que indique si el usuario está activo. A continuación, se procederá a configurar Spring Security, para lo que se creará una nueva clase en la que se pueden establecer múltiples opciones, como por ejemplo qué páginas se desbloquean para cada rol, la página de login personalizada, un codificador de contraseña, entre otras.

Una vez se tiene la clase correctamente configurada, el siguiente paso es realizar la página de login personalizada y comprobar que inicia correctamente sesión con los usuarios (véase Figura 4.19). Luego es necesario volver a trabajar con Thymeleaf en las vistas ya creadas, en las que hay que ocultar ciertos botones y funcionalidades de la aplicación dependiendo del rol y del usuario. Por ejemplo, el hecho de que una noticia solo la pueda modificar el periodista que la creó o un administrador, o que se muestre el botón de desconexión (log out) cuando se está autenticado en la página.

Una vez se ha concluído con el sistema de usuarios, se puede dar esta última iteración por finalizada.

4.5.1 Observaciones

Con esta última iteración, se obtiene una primera versión completa de la aplicación Web tras la inclusión del sistema de usuarios.

En términos de dificultad, resultó ser más complicada de lo esperado dado que la configuración de Spring Security era bastante diferente a lo visto anteriormente, por lo que la fase de análisis e información de las tecnologías a usar resultó ser un poco más larga de lo previsto. Sin embargo, en la parte de personalización de las vistas no hubo problemas relevantes gracias a la experiencia obtenida en las iteraciones anteriores.

El siguiente paso tras la finalización de la aplicación sería su mantenimiento, solucionando

los fallos que se vayan encontrando o añadiendo nuevas funcionalidades y características útiles que pudiera solicitar un potencial cliente. Para ello, en el siguiente capítulo se explica en detalle el despliegue utilizado para que esta fase también se pueda realizar de una manera óptima y eficiente.

Capítulo 5

Despliegue de la aplicación Web

Una vez desarrollada la aplicación Web, llega el momento de su despliegue en la nube pública. Durante este proceso de despliegue usaremos principalmente tres tecnologías:

- **Kubernetes:** Herramienta que nos permitirá orquestar un clúster de contenedores en el cual se ejecutará nuestra aplicación.
- **Terraform:** Herramienta mediante la cual crearemos el clúster Kubernetes en la nube pública de una manera simple y automatizada, gracias al paradigma IaC.
- **Amazon Web Services (AWS):** Proveedor de servicios de computación en la nube, el cual será configurado mediante Terraform para desplegar el clúster Kubernetes usando su servicio gestionado Amazon EKS.

Utilizando la misma metodología que en la parte de desarrollo, este proceso de despliegue consiste en dos iteraciones en este caso. En ellas se distingue entre el diseño del clúster con Kubernetes desplegado en un entorno local y su posterior migración al servicio en la nube EKS utilizando Terraform.

5.1 Primera iteración: Diseño del clúster Kubernetes

5.1.1 Descripción e implementación

Primeramente, se deben sustituir en el archivo *application.properties* de la aplicación Web los valores correspondientes a la conexión con la base de datos por variables del sistema. Estas variables serán declaradas al ejecutar la aplicación en el contenedor. Tras eso, se genera el archivo ".jar" con la herramienta Maven y se crea una imagen Docker con él, utilizando una imagen base que tenga Java 14 instalado o bien instalándolo mediante comandos (en nuestro caso se ha utilizado la primera opción, mostrada en el Listing 5.1). Dicha imagen hay que

```
1 FROM adoptopenjdk/openjdk14:alpine-jre
2 ADD App-0.0.1-SNAPSHOT.jar app.jar
3 ENTRYPOINT ["java","-jar","app.jar"]
```

Listing 5.1: Dockerfile usado para crear la imagen Docker

subirla al repositorio público DockerHub, desde donde será descargada posteriormente para desplegarla en el clúster Kubernetes.

Seguidamente, se procede con la instalación de un clúster Kubernetes en la máquina local. En este caso, se ha seguido una guía de HashiCorp (el desarrollador de Terraform) para instalarlo en un sistema operativo Ubuntu mediante "Kind" [25]. Una vez instalado, procederemos a informarnos acerca de los distintos tipos de elementos existentes en Kubernetes (explicados en la Sección 2.2.2) para poder generar el fichero YAML que representa el despliegue a realizar (o varios, si se quiere separar para una mejor organización), y en el cual se irán añadiendo según hagan falta. A continuación, se declaran los elementos presentes en dicho fichero YAML, junto con una explicación de la función realizada por cada uno de ellos:

- Secrets: Utilizados para pasar las contraseñas de la base de datos y del repositorio DockerHub.
- ConfigMaps: Usados para pasar el script de la base de datos al Deployment de MySQL y la configuración de la conexión con la base de datos al Deployment de la aplicación.
- Volumes: Un volúmen será el encargado de almacenar la información de la base de datos, y otro de guardar las imágenes que se suben a la aplicación Web.
- Deployments: Su función es declarar el estado deseado de los despliegues de la base de datos MySQL y de la aplicación Web. Es posiblemente el tipo de elemento más complejo de todos, pues es donde se realizarán todas las conexiones con la mayoría de los elementos listados hasta ahora, además de especificar que imagen se usa, mínimo y máximo de CPU, numero de réplicas, etc (véase el fragmento de código 5.2). Cabe destacar que, como buena práctica, se recomienda establecer el uso de CPU máximo por debajo de un núcleo (core) siempre que el despliegue o aplicación no estén específicamente diseñados para trabajar con varios núcleos. Si se desea mayor capacidad de computación, siempre se puede aumentar el número de réplicas a utilizar en el despliegue.
- Services: Un servicio será responsable de exponer la base de datos a la aplicación Web, y el otro de exponer la propia aplicación al exterior. En el entorno actual, se utilizará el tipo de servicio Kubernetes denominado "NodePort" para ambos, además de configurar la afinidad de sesiones para el servicio de la aplicación (la cual permitirá mantener la sesión en caso de haber más de una réplica).

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: opinews
spec:
  selector:
    matchLabels:
      app: opinews
  template:
    metadata:
      labels:
        app: opinews
    spec:
      containers:
        - name: opinews
          image: nico345/opinews:1.1
          imagePullPolicy: Always
          ports:
            - containerPort: 8080
          resources:
            requests:
              memory: "64Mi"
              cpu: "0.15"
            limits:
              memory: "264Mi"
              cpu: "0.3"
          env:   # Variables
          - name: DB_NAME
            valueFrom:
              configMapKeyRef:
                name: opinewsdb-config
                key: database
          - name: DB_HOST
            valueFrom:
              configMapKeyRef:
                name: opinewsdb-config
                key: host
          - name: DB_PASSWORD
            valueFrom:
              secretKeyRef:
                name: opinews-secret
                key: db-password
          volumeMounts:
          - name: opinews-images
            mountPath: /opinews/img-noticias
      volumes:
      - name: opinews-images
        persistentVolumeClaim:
          claimName: opinewsimg-pv-claim
      imagePullSecrets:
        - name: dockerhub
```

Listing 5.2: Deployment de la aplicación

```
sudo kubectl top pods       #mostrar metricas
NAME                         CPU(cores)   MEMORY(bytes)
opinews-6b6b4fdc95-6rd92     2m           195Mi
opinewsdb-7b9cc8c6bc-c9rlz   3m           246Mi

sudo kubectl get hpa        #mostrar autoescalado
NAME       REFERENCE              TARGETS    MINPODS   MAXPODS
    REPLICAS   AGE
opinews    Deployment/opinews   1%/95%     1         3
    1          5d
```

Listing 5.3: Comprobación del servidor de métricas y el autoescalado en Ubuntu

Al aplicar dicha configuración al clúster Kubernetes mediante el comando kubectl, se debería obtener la página funcionando correctamente en el entorno local. Una vez llegados a este punto, se procede a desplegar en el clúster un servidor de métricas, el cual nos permitirá conocer cuánta CPU y memoria están usando nuestros contenedores. Esto será necesario para posteriormente programar un autoescalado horizontal, el cual aumentará el número de réplicas cuando el uso de CPU alcance un determinado porcentaje. Cuando las métricas funcionen correctamente, es posible probarlas tal y como se muestra en el fragmento de código 5.3, donde primero se obtiene el uso de CPU y memoria actual de los Pods (kubectl top pods) y a continuación de comprueba el estado del autoescalado horizontal (kubectl get hpa).

5.1.2 Observaciones

Esta iteración resultó un poco más complicada que las anteriores de desarrollo pues, a pesar del interés del alumno en Kubernetes, no se había configurado ni desplegado nunca un proyecto similar con este entorno. Sin embargo, se ha conseguido realizar la iteración sin problemas demasiado complicados gracias a la cantidad de información y tutoriales que existen en la Web sobre Kubernetes.

5.2 Segunda iteración: Despliegue en la nube pública con Terraform

5.2.1 Descripción e implementación

Una vez se dispone de un clúster Kubernetes funcional en la máquina local y se comprende su funcionamiento, el siguiente paso consiste en replicarlo en el servicio EKS mediante Terraform. Al igual que en las otras iteraciones del proyecto, primero existe una etapa de formación acerca de cómo funciona Terraform y de todo lo necesario para desplegar un clúster Kubernetes en EKS.

```
#VPC
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "2.6.0"

  name                 = "k8s-vpc"
  cidr                 = "172.16.0.0/16"
  azs                  = data.aws_availability_zones.available.names
  private_subnets      = ["172.16.1.0/24", "172.16.2.0/24",
    "172.16.3.0/24"]
  public_subnets       = ["172.16.4.0/24", "172.16.5.0/24",
    "172.16.6.0/24"]
  enable_nat_gateway   = true
  single_nat_gateway   = true
  enable_dns_hostnames = true

  public_subnet_tags = {
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
    "kubernetes.io/role/elb"                    = "1"
  }

  private_subnet_tags = {
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
    "kubernetes.io/role/internal-elb"           = "1"
  }
}
```

Listing 5.4: VPC declarada en el fichero de Terraform

Antes de nada, es necesario realizar la autenticación en AWS desde Terraform, por lo que debemos obtener nuestras credenciales para así poder realizar las operaciones en nuestra cuenta. Esto se puede hacer de diferentes formas (algunas varían en función del sistema operativo), pero al estar desarrollando esta parte en Ubuntu se ha decidido instalar la herramienta awscli y utilizar el comando "aws configure" [26].

También será necesario configurar una Virtual Private Cloud (VPC) en la cual se desplegarán las instancias EC2 encargadas de ejecutar el clúster Kubernetes y la aplicación. Dichos servicios también serán configurados desde Terraform gracias al proveedor de AWS, para lograr así una mayor automatización. En el fragmento de código 5.4 se puede observar la VPC declarada en el fichero de Terraform.

Con la VPC ya implementada en código, el siguiente paso es declarar el módulo EKS para que Terraform sepa como trabajar con dicho servicio, además del proveedor de Kubernetes [27] y un módulo para el servidor de métricas, los cuales se usarán para replicar en EKS la configuración realizada en la iteración anterior.

Tras realizar estos pasos y tener la configuración replicada de la iteración anterior en el fichero de Terraform, es necesario realizar ciertos cambios para lograr un funcionamiento correcto y óptimo. Para empezar, se debe cambiar el servicio encargado de exponer la aplica-

ción. Al estar en un entorno como AWS, es posible utilizar el tipo de servicio LoadBalancer, el cual distribuirá el tráfico entrante de forma más eficiente entre varios contenedores y que, al mostrar los servicios desde terminal, devolverá una URL a la que acceder desde el navegador. Sin embargo, no es posible configurar una afinidad de sesión en este tipo de servicios, por lo que se tendría que desplegar un controlador Ingress.

Dicho controlador Ingress, además de realizar también un balanceo de la carga, nos permite utilizar ciertas funciones útiles como comprobar el estado de la página, exponer varios entornos con la misma URL (p.e. añadiendo "/prueba" podríamos tener un despliegue con una versión en la que los desarrolladores realicen pruebas mientras que en "/public" estaría la actual) o establecer la afinidad de sesión, entre otras muchas. Para instalar el controlador se utilizará Helm [28], una herramienta que permite desplegar ciertos programas o utilidades en un clúster Kubernetes mediante la utilización de plantillas, simplificando el proceso. Para ello, es necesario utilizar el proveedor de Helm e indicarle la URL de la plantilla que queremos usar junto con las variables necesarias (región de AWS, nombre del clúster, etc). En este caso se ha decidido usar el controlador Application Load Balancer (ALB) [29] que proporciona AWS, tras lo cual procederemos a configurarlo de una manera similar al resto de elementos de Kubernetes, indicándole en las opciones la afinidad de sesión (mostrado en el fragmento de código 5.5).

A continuación, una vez añadido el controlador Ingress se procede a establecer una estrategia de actualización de la aplicación Web denominada "zero-downtime rolling update", mediante la cual Kubernetes mantendrá un número de réplicas funcionales con la versión anterior de la aplicación hasta que tenga actualizadas las réplicas con la nueva versión, eliminando así posibles tiempos de caída al realizar una actualización de la aplicación [30]. La importancia del controlador Ingress en esta estrategia de actualización es su función para poder comprobar el estado de la página. Gracias a ella es posible evitar que Kubernetes cambie de la versión antigua a la nueva antes de que el contenedor tenga tiempo a iniciar la aplicación Web, evitando así errores al acceder a la URL.

Para terminar, faltaría desplegar la aplicación con Terraform, lo cual consiste en tres sencillos pasos:

- **Inicializar Terraform:** Durante el transcurso del despliegue se ha estado mencionando la inserción de ciertos módulos y proveedores mediante los cuales es posible configurar las distintas tecnologías de las que hacemos uso. Sin embargo, Terraform no viene con estos elementos instalados de serie, por lo que se necesitaría realizar un "terraform init" para así descargarlos e instalarlos antes de poder usarlos [31].

- **Planear los cambios:** Con Terraform configurado, se procede a ejecutar un "terraform plan", lo que devuelve un resumen de los cambios que va a realizar la herramienta en el

```
#Controlador ingress
resource "kubernetes_ingress" "ingress" {
  metadata {
    name = "ingress"
    annotations={
      "kubernetes.io/ingress.class" = "alb"
      "alb.ingress.kubernetes.io/target-type"= "ip"
      "alb.ingress.kubernetes.io/target-group-attributes" =
            "stickiness.enabled=true,
             stickiness.lb_cookie.duration_seconds=1200"
      "alb.ingress.kubernetes.io/scheme" = "internet-facing"
    }
  }
  spec {
    rule {
      http {
        path {
          backend {
            service_name = "opinews"
            service_port = 80
          }

          path = "/*"
        }
      }
    }
  }
}
```

Listing 5.5: Configuración del controlador Ingress

entorno objetivo (crear clúster Kubernetes, crear VPC, etc), además de indicar posibles fallos de sintaxis [32].

- **Aplicar los cambios:** Por último, se ejecuta el comando "terraform apply", que hace efectivos los cambios planificados anteriormente. Este comando puede tardar en completarse un tiempo significativo (15-20 min), pues es en este paso cuando finalmente se aprovisiona toda la infraestructura necesaria en la nube [33].

Una vez realizados los pasos anteriores, ya disponemos de la aplicación desplegada en un clúster de contenedores en la nube el cual, a diferencia de un servidor convencional, nos permite escalar los recursos en caso de alta demanda y eliminarlos en caso contrario. Además, es posible realizar actualizaciones de la aplicación sin que haya cortes en el servicio mediante la técnica "zero-downtime rolling update". Por último, gracias al uso del paradigma IaC mediante Terraform, la aplicación se puede desplegar de forma sencilla y automatizada sin necesidad de realizar cambios en su estructura ni tener que crearla desde cero (aunque esto sí podría suponer un corte del servicio).

5.2.2 Observaciones

Esta iteración llevó más tiempo de lo estimado inicialmente debido a que principalmente se trabajó con una cuenta de AWS Educate hasta que se descubrió que esta tenía ciertas restricciones de uso en el servicio de "OpenID Connect Provider". Aunque este servicio no requiere ser modificado activamente, sí evitaba el correcto despliegue del controlador Ingress a pesar de que estaba bien configurado. Hubo que realizar el cambio a una cuenta AWS estándar (sin restricciones), y entonces el controlador Ingress funcionó sin problemas.

Respecto al trabajo con Terraform y EKS, aunque no supuso mucha dificultad desplegar el servicio sí que resultó difícil de comprender, pues en la carrera apenas se había hecho una práctica con AWS y nunca antes se había trabajado con una herramienta parecida a Terraform. Sin embargo, una vez se tenía la estructura desplegada, la replicación del clúster y los cambios necesarios no supusieron demasiado problema (exceptuando el indicado anteriormente).

Capítulo 6

Conclusiones

Al terminar el proyecto se puede afirmar que se han logrado cumplir los objetivos propuestos inicialmente: el desarrollo de una aplicación Web y su despliegue automatizado en la nube pública mediante el paradigma IaC. La realización de este trabajo me ha permitido agrupar conocimientos de varias asignaturas del grado como Programación Integrativa, Administración de Infraestructuras y Sistemas Informáticos, Bases de Datos e Integración de Datos (entre otras) con otros aprendidos a lo largo de este trabajo para crear así un producto software completo que hace uso de tecnologías y metodologías muy utilizados y populares en la actualidad (Java, Kubernetes, Terraform, plataformas de computación en la nube, etc). El resultado del trabajo realizado en este TFG se encuentra disponible públicamente en el siguiente repositorio Git: https://github.com/Nico345/TFG.

Pese a la falta de experiencia y conocimiento con la mayoría de tecnologías usadas, su elección se puede considerar muy acertada, pues dichas herramientas nos han proporcionado funciones y métodos muy útiles que han permitido progresar de manera más sencilla y rápida. Especialmente en el caso de Spring y sus dependencias, las cuales nos han facilitado enormemente el trabajo de implementar las páginas y la conexión con la base de datos. Gracias a esto hemos podido acortar la duración del trabajo a pesar del problema que surgió en la segunda iteración del despliegue con la puesta en marcha del controlador Ingress. Este hecho provocó un ligero retraso con respecto a la planificación inicial, lo que se refleja en el cronograma final mostrado en la Figura 6.1.

En un ámbito más personal, la realización del proyecto proporcionó al alumno experiencia y mayor conocimiento en el campo de DevOps, en el cual está interesado y en el que ha experimentado una mejora al realizar tanto el desarrollo como el despliegue, lo que le permitirá acortar el ciclo de vida del desarrollo de sistemas o aplicaciones y proporcionar una entrega continua con software de alta calidad.

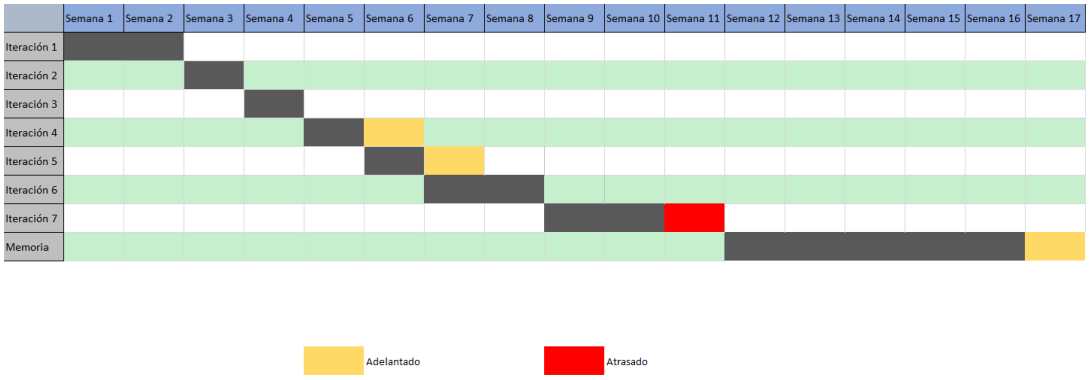

Figura 6.1: Cronograma actualizado

6.1 Trabajo futuro

Aunque se ha conseguido el objetivo principal de realizar el desarrollo completo de una aplicación y desplegarla en la nube de forma que proporcione escalabilidad y alta disponibilidad, quedan algunos puntos que se podrían mejorar del trabajo en ciclos futuros, como por ejemplo:

- Ahora mismo no se está haciendo uso de un certificado Secure Sockets Layer (SSL) para validar la página, por lo que puede aparecer como insegura. En el futuro se podría configurar el controlador Ingress para que use uno.
- Se podría crear otra base de datos y programar la realización de backups automáticos para así no perder todos los datos en caso de que la actual falle (lo que se podría aplicar también en el almacenamiento de las imágenes).

A mayores, también sería interesante aumentar las funcionalidades que proporciona la aplicación ahora mismo para incorporar otras útiles que no están implementadas en esta primera versión, como la recuperación de la contraseña, permitir a un administrador cambiar el rol de un usuario desde la aplicación (ahora mismo es necesario cambiarlo desde base de datos), o crear página personalizadas para los errores como el 404. Cabe destacar que las nuevas funcionalidades (o la corrección de posibles bugs que puedan surgir) se podrían aplicar sin cortes en el servicio, gracias a la técnica “zero-downtime rolling update” que se ha implementado en la fase de despliegue.

Apéndices

Apéndice A

Archivos YAML de configuración de Kubernetes

En este apéndice se muestra el contenido de los ficheros YAML desplegados en el clúster local de Kubernetes, que posteriormente son replicados en EKS mediante Terraform. En el fichero A.1 se define un Secret con la contraseña del repositorio DockerHub desde donde se descargará la imagen Docker, mientras que en el fichero A.2 se muestra otro Secret con la contraseña de la base de datos codificada en base64, la cual será necesaria para conectarse desde la aplicación Web.

En el fichero A.3 se define el ConfigMap utilizado para pasar el script que genera la estructura de la base de datos y añade tres usuarios para probarla, mientras que el ConfigMap que define el host y el nombre de la base de datos que le pasaremos a la aplicación Web se muestra en el fichero A.4.

En el fichero A.5 hay tres elementos declarados:

- Service que expondrá la base de datos para que la aplicación Web pueda conectarse.
- PersistentVolumeClaim que aprovisionará el PersistentVolume donde se almacenará la información de la base de datos.
- Deployment donde especificaremos la imagen usada, cantidad de CPU y memoria, así como las variables necesarias para crear un contenedor con la base de datos.

El último fichero (A.6) sigue la misma estructura que el anterior, definiendo otros tres elementos:

- Service que expondrá la aplicación Web para poder conectarse desde el exterior.
- PersistentVolumeClaim que aprovisionará el PersistentVolume donde se almacenarán las imágenes que se suban desde la aplicación.

- Deployment donde especificaremos la imagen usada, cantidad de CPU y memoria, así como las variables necesarias para crear un contenedor con la aplicación.

```
apiVersion: v1
kind: Secret
metadata:
  name: dockerhub
data:
  .dockerconfigjson:
    ewoJImF1dGhzIjogewoJCSJodHRwczovL2luZGV4LmRvY2tlci5pby92MS8iOiB7C
    gkJCSJhdXRoIjogImJtbGpiek0wTlRwVVlXMWhiV0UyTmpZMk5qWT0iCgkJfQoJf
    Qp9
type: kubernetes.io/dockerconfigjson
```

Listing A.1: dockerhub-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: opinews-secret
type: Opaque
data:
  db-password: cGFzc3dvcmQ=
```

Listing A.2: opinews-secret.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: opinews-initdb-config
data:
  opinewsdb.sql:
    USE opinewsdb;

    CREATE TABLE IF NOT EXISTS `Usuarios` (
      `idUsuario` int(11) NOT NULL AUTO_INCREMENT,
      `nickname` varchar(45) NOT NULL,
      `email` varchar(100) NOT NULL,
      `rol` varchar(45) NOT NULL,
      `password` varchar(100) NOT NULL,
      `estado` tinyint NOT NULL,
      PRIMARY KEY (`idUsuario`),
      UNIQUE KEY `nickname_UNIQUE` (`nickname`),
      UNIQUE KEY `email_UNIQUE` (`email`)
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

    CREATE TABLE IF NOT EXISTS `Temas` (
```

```
22        `idTema` int(11) NOT NULL AUTO_INCREMENT,
23        `categoria` varchar(45) NOT NULL,
24        `nombre` varchar(100) NOT NULL,
25        PRIMARY KEY (`idTema`),
26        UNIQUE KEY `tema_UNIQUE` (`nombre`,`categoria`)
27      ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
28
29
30      CREATE TABLE IF NOT EXISTS `Noticias` (
31        `idNoticia` int(11) NOT NULL AUTO_INCREMENT,
32        `fecha` date NOT NULL,
33        `usuario` int(11) NOT NULL,
34        `vistas` int(11) NOT NULL,
35        `titulo` varchar(200) NOT NULL,
36        `subtitulo` varchar(300) NOT NULL,
37        `texto` text NOT NULL,
38        `tema` int(11) NOT NULL,
39        `imagen` varchar(100),
40        PRIMARY KEY (`idNoticia`),
41        CONSTRAINT `fk_temas_noticias` FOREIGN KEY (`tema`)
        REFERENCES `Temas` (`idTema`),
42        CONSTRAINT `fk_usuarios_noticias` FOREIGN KEY (`usuario`)
        REFERENCES `Usuarios` (`idUsuario`)
43      ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
44
45      CREATE TABLE IF NOT EXISTS `Posts` (
46        `idPost` int(11) NOT NULL AUTO_INCREMENT,
47        `fecha` date NOT NULL,
48        `usuario` int(11) NOT NULL,
49        `vistas` int(11) NOT NULL,
50        `titulo` varchar(200) NOT NULL,
51        `texto` text NOT NULL,
52        `tema` int(11) NOT NULL,
53        `imagen` varchar(100),
54        PRIMARY KEY (`idPost`),
55        CONSTRAINT `fk_temas_posts` FOREIGN KEY (`tema`) REFERENCES
        `Temas` (`idTema`),
56        CONSTRAINT `fk_usuarios_posts` FOREIGN KEY (`usuario`)
        REFERENCES `Usuarios` (`idUsuario`)
57      ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
58
59      CREATE TABLE IF NOT EXISTS `Comentarios` (
60        `idComentario` int(11) NOT NULL AUTO_INCREMENT,
61        `fecha` date NOT NULL,
62        `usuario` int(11) NOT NULL,
63        `post` int(11) NOT NULL,
```

```
64      `texto` text NOT NULL,
65      PRIMARY KEY (`idComentario`),
66      CONSTRAINT `fk_posts_comentarios` FOREIGN KEY (`post`)
    REFERENCES `Posts` (`idPost`),
67      CONSTRAINT `fk_usuarios_comentarios` FOREIGN KEY (`usuario`)
    REFERENCES `Usuarios` (`idUsuario`)
68    ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
69
70    INSERT INTO Usuarios(nickname,email,rol,password,estado) VALUES
    ("user","user@mail.com","usuario","\$2a\$10\$rJLa8yVy9B72wJOm2zrp
    WOA9sTuNLlieJp88jOPfUEQjHvZety2HW",1);
71    INSERT INTO Usuarios(nickname,email,rol,password,estado) VALUES
    ("periodista","periodista@mail.com","periodista","\$2a\$10\$dgeMz
    Krrd/VuF2rR5awNQeK54hIBO5JdFtVvPkgJyhmMKNAFM1KhC",1);
72    INSERT INTO Usuarios(nickname,email,rol,password,estado) VALUES
    ("admin","admin@mail.com","admin","\$2a\$10\$7TU4FDSRK9TwQUMQz0MC
    d.26aFl16H9NbQmqh0LIW9dWUR9SFxFD6",1);
```

Listing A.3: script.yaml

```
1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: opinewsdb-config
5  data:
6    host: opinewsdb
7    database: opinewsdb
```

Listing A.4: opinewsdb-configmap.yaml

```
1  #Servicio de Mysql
2  apiVersion: v1
3  kind: Service
4  metadata:
5    name: opinewsdb
6    labels:
7      app: opinewsdb
8      tier: database
9  spec:
10   ports:
11     - port: 3306
12       targetPort: 3306
13   selector:
14     app: opinewsdb
15     tier: database
16   clusterIP: None #No hace falta al usar DNS
17 ---
```

```yaml
#Persistent Volume Claim para almacenar los datos de Mysql
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: opinewsdb-pv-claim
  labels:
    app: opinewsdb
    tier: database
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
---
#Deployment de Mysql
apiVersion: apps/v1
kind: Deployment
metadata:
  name: opinewsdb
  labels:
    app: opinewsdb
    tier: database
spec:
  selector:
    matchLabels:
      app: opinewsdb
      tier: database
  template:
    metadata:
      labels:
        app: opinewsdb
        tier: database
    spec:
      containers:
      - name: opinewsdb
        image: mysql
        ports:
        - containerPort: 3306
          name: opinewsdb
        resources:
          requests:
            memory: "64Mi"
            cpu: "0.15"
          limits:
            memory: "264Mi"
```

```
64              cpu: "0.3"
65          env:    #Variables
66          - name: MYSQL_ROOT_PASSWORD # Variable de la contrasena de
    la BDD
67            valueFrom:
68              secretKeyRef:
69                name: opinews-secret
70                key: db-password
71          - name: MYSQL_DATABASE # Nombre BDD
72            value: opinewsdb
73          volumeMounts:
74          - name: opinewsdb-storage # Volumen de almacenamiento
75            mountPath: /var/lib/mysql
76          - name: opinews-initdb-config # Script para crear BDD
    inicial
77            mountPath: /docker-entrypoint-initdb.d
78        volumes:
79        - name: opinewsdb-storage
80          persistentVolumeClaim:
81            claimName: opinewsdb-pv-claim
82        - name: opinews-initdb-config
83          configMap:
84            name: opinews-initdb-config
```

Listing A.5: opinewsdb-deployment.yaml

```
1  #Servicio Opinews
2  apiVersion: v1
3  kind: Service
4  metadata:
5    name: opinews
6    labels:
7      name: opinews
8  spec:
9    ports:
10     - nodePort: 30163
11       port: 8080
12       targetPort: 8080
13       protocol: TCP
14   selector:
15     app: opinews
16   type: NodePort
17   sessionAffinity: ClientIP
18 ---
19 #Persistent Volume Claim para almacenar las imágenes
20 apiVersion: v1
```

```
kind: PersistentVolumeClaim
metadata:
  name: opinewsimg-pv-claim
  labels:
    app: opinews
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
---
#Deployment Opinews
apiVersion: apps/v1
kind: Deployment
metadata:
  name: opinews
spec:
  selector:
    matchLabels:
      app: opinews
  template:
    metadata:
      labels:
        app: opinews
    spec:
      containers:
        - name: opinews
          image: nico345/opinews:1.1
          imagePullPolicy: Always
          ports:
            - containerPort: 8080
          resources:
            requests:
              memory: "64Mi"
              cpu: "0.15"
            limits:
              memory: "264Mi"
              cpu: "0.3"
          env:   # Variables
          - name: DB_NAME  # Estableciendo BDD desde ConfigMap
            valueFrom:
              configMapKeyRef:
                name: opinewsdb-config
                key: database
          - name: DB_HOST  # Estableciendo host de la BDD desde
```

```
    ConfigMap
67            valueFrom:
68              configMapKeyRef:
69                name: opinewsdb-config
70                key: host
71          - name: DB_PASSWORD # Estableciendo contrasena de la BDD
    desde Secret
72            valueFrom:
73              secretKeyRef:
74                name: opinews-secret
75                key: db-password
76          volumeMounts: # Montando volumen de almacenamiento de las
    imagenes
77          - name: opinews-images
78            mountPath: /opinews/img-noticias
79        volumes:
80        - name: opinews-images
81          persistentVolumeClaim:
82            claimName: opinewsimg-pv-claim
83        imagePullSecrets: # Contrasena del repositorio
84          - name: dockerhub
```

Listing A.6: opinews-deployment.yaml

Lista de acrónimos

ALB Application Load Balancer. 34

AWS Amazon Web Services. 7–9, 11, 13, 29, 33, 34, 36

BSc Bachelor of Science. 1

CRUD Create-Read-Update-Delete. 12, 13, 23

EC2 Elastic Compute Cloud. 7, 33

EKS Elastic Kubernetes Service. 1, 2, 8, 13, 29, 32, 33, 36, 41

IaC Infrastructure as Code. 1, 2, 5, 8, 13, 29, 35, 37

JDK Java Development Kit. 11

JPA Java Persistence Api. iii, 4, 23, 24

JVM Java Virtual Machine. 3

MVC Modelo-Vista-Controlador. 1, 4

SQL Structured Query Language. 4

SSL Secure Sockets Layer. 38

TFG Trabajo de Fin de Grado. 1, 2, 4, 6–9, 37

VPC Virtual Private Cloud. vii, 8, 33, 35

Bibliografía

[1] K. Morris, *Infrastructure as Code: Managing servers in the cloud.* O'Reilly Media, Inc., 2016.

[2] "Web MVC framework." [En línea]. Disponible en: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html

[3] "Spring Data JPA." [En línea]. Disponible en: https://spring.io/projects/spring-data-jpa

[4] "Spring Security." [En línea]. Disponible en: https://spring.io/projects/spring-security

[5] "Maven." [En línea]. Disponible en: https://maven.apache.org/

[6] "Tutorial: Using thymeleaf." [En línea]. Disponible en: https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html

[7] "Bootstrap: The most popular HTML, CSS and JS library in the world." [En línea]. Disponible en: https://getbootstrap.com/

[8] "MySQL." [En línea]. Disponible en: https://www.mysql.com/

[9] D. Bernstein, "Containers and cloud: From LXC to Docker to Kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[10] "Deployment | kubernetes." [En línea]. Disponible en: https://kubernetes.io/es/docs/concepts/workloads/controllers/deployment/

[11] "Pods | Kubernetes." [En línea]. Disponible en: https://kubernetes.io/es/docs/concepts/workloads/pods/pod/

[12] "Configmap | kubernetes." [En línea]. Disponible en: https://kubernetes.io/es/docs/concepts/configuration/configmap/

[13] "Volumes | kubernetes." [En línea]. Disponible en: https://kubernetes.io/docs/concepts/storage/volumes/

[14] "Secret | kubernetes." [En línea]. Disponible en: https://kubernetes.io/es/docs/concepts/configuration/secret/

[15] "Persistent Volumes | Kubernetes." [En línea]. Disponible en: https://kubernetes.io/docs/concepts/storage/persistent-volumes/

[16] "Service | kubernetes." [En línea]. Disponible en: https://kubernetes.io/docs/concepts/services-networking/service/

[17] A. W. S. (AWS), "Página principal de aws." [En línea]. Disponible en: https://aws.amazon.com/es/

[18] "AWS | Elastic Compute Cloud (EC2)," 2021. [En línea]. Disponible en: https://aws.amazon.com/es/ec2/?ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc

[19] "Elastic kubernetes services." [En línea]. Disponible en: https://aws.amazon.com/es/eks/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc&eks-blogs.sort-by=item.additionalFields.createdDate&eks-blogs.sort-order=desc

[20] "AWS | Red virtual privada en la nube (VPC)." [En línea]. Disponible en: https://aws.amazon.com/es/vpc/?vpc-blogs.sort-by=item.additionalFields.createdDate&vpc-blogs.sort-order=desc

[21] "¿Qué es un Application Load Balancer?" [En línea]. Disponible en: https://docs.aws.amazon.com/es_es/elasticloadbalancing/latest/application/introduction.html

[22] "Terraform." [En línea]. Disponible en: https://www.terraform.io/

[23] "Docker." [En línea]. Disponible en: https://www.docker.com/

[24] "Visual Studio Core." [En línea]. Disponible en: https://code.visualstudio.com/

[25] "Kind installation." [En línea]. Disponible en: https://kind.sigs.k8s.io/docs/user/quick-start#installation

[26] "Opciones de los archivos de configuración y credenciales." [En línea]. Disponible en: https://docs.aws.amazon.com/es_es/cli/latest/userguide/cli-configure-files.html

[27] "Docs overview hashicorp/kubernetes." [En línea]. Disponible en: https://registry.terraform.io/providers/hashicorp/kubernetes/latest/docs

[28] "Helm." [En línea]. Disponible en: https://helm.sh/

[29] “Balancing de carga de aplicaciones en Amazon EKS.” [En línea]. Disponible en: https://docs.aws.amazon.com/es_es/eks/latest/userguide/alb-ingress.html

[30] “Performing a Rolling Update | Kubernetes.” [En línea]. Disponible en: https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/

[31] “Initializing Working Directories.” [En línea]. Disponible en: https://www.terraform.io/docs/cli/init/index.html

[32] “Command: plan.” [En línea]. Disponible en: https://www.terraform.io/docs/cli/commands/plan.html

[33] “Command: apply.” [En línea]. Disponible en: https://www.terraform.io/docs/cli/commands/apply.html