MASKDADOS.com, la web de ayuda a los juegos de rol desarrollada en Google Cloud Platform usando CICD con Jenkins y aplicando la escalabilidad de Kubernetes.

Ronny Romero Navarrete

Resumen– En este artículo queda reflejado todo el proceso de desarrollo de una aplicación web utilizando las tecnologías actuales de Cloud Computing. El proyecto construye las bases de una plataforma online que ayuda a los jugadores a disponer de una herramienta extra para sus partidas. Esta herramienta consiste en una página web de ayuda a los juegos de rol de mesa, incorporando un sistema de login y gestión personal de fichas de personaje. Con esa finalidad, el proyecto se ha desarrollado aplicando la metodología CICD a lo largo de todo el proceso. A través de la plataforma de Google Cloud Platform se ha creado toda la infraestructura y los servicios necesarios para operar en la nube, terminando con un despliegue mediante el uso de Kubernetes y el beneficio de su escalabilidad.
Palabras clave– Google Cloud Platform, Cloud Computing, GCP, Jenkins, CICD, Kubernetes, Google Kubernetes Engine, Docker, Spring Boot, Angular, JWT, rol, web, juego.

Abstract– This article reflects the entire process of developing a web application using current Cloud Computing technologies. The project builds the foundations of an online platform that helps players to have an extra tool for their games. The application consists of a web page to help tabletop role-playing games, incorporating a personal login system and management of character sheets. To that end, the project has been developed by applying the CICD methodology throughout the entire process. Through the Google Cloud Platform, all the necessary infrastructure and services have been created to operate in the cloud, ending with a deployment through the use of Kubernetes and the benefits of its scalability.

Keywords– Google Cloud Platform, Cloud Computing, GCP, Jenkins, CICD, Kubernetes, Google Kubernetes Engine, Docker, Spring Boot, Angular, JWT, role, web, game.

1 Introducción

Este proyecto tratará sobre el desarrollo de una aplicación web de gestión de ayuda a los juegos de rol en mesa, utilizando las ventajas de la computación en la nube, aplicando la metodología CICD en todo el proceso de desarrollo y aprovechando la escalabilidad que ofrece Kubernetes para el despliegue.
La elección del tema de proyecto ha surgido de la idea de crear una aplicación real utilizando tecnologías actuales, inspirado por las últimas tendencias en el desarrollo. Por ello, el trabajo realizado en este informe se divide en cuatro partes: Computación en la nube, CICD, Kubernetes y el desarrollo de una aplicación web utilizando Angular y Spring Boot.

• E-mail de contacte: ronnyrom3@gmail.com
• Menció realitzada: Enginyeria del Software
• Treball tutoritzat per: Fernando Vilariño
• Curs 2019/20

2 Objetivos

Como se ha comentado en la introducción, este proyecto consiste en crear una aplicación web de ayuda a los juegos de rol de mesa, utilizando la infraestructura y servicios de Google Cloud aplicando la metodología CICD para desplegar la aplicación utilizando Kubernetes. Para conseguirlo se han establecido 4 objetivos:

- Montar la arquitectura en Google Cloud Platform utilizando las diferentes herramientas que ofrece su nube para crear la infraestructura y servicios que necesitará el proyecto.
- Aplicar la metodología CICD (*Continous Integration + Continuous Delivery + Continuos Deployment*) utilizando Jenkins para poder automatizar todo el proceso que va desde la entrega de software hasta el despliegue en Kubernetes de Google Cloud con todas sus fases.
- Utilizar la herramienta de Google Kubernetes Engine para desplegar toda la aplicación en ella pudiendo aprovechar las escalabilidad que ofrece Kubernetes.
- Desarrollar una aplicación web de ayuda para los juegos de rol utilizando una arquitectura de micro servicios API REST separando claramente la parte del *front end* y la parte del *back end*.

3 Estado del arte

En la actualidad las tecnologías con las que se trabajará en este proyecto son cada vez más usadas.

3.1. Cloud computing

Cada vez más, las empresas comienzan a dejar de usar su propia infraestructura para el despliegue o desarrollo de aplicaciones con la intención de externalizarlo hacia el "Cloud computing". Delegan toda esta parte a terceras empresas que operan en la nube que se encargarán de gestionarlo todo. Actualmente, junto con Google Cloud Platform hay muchas empresas que que se dedican a ofrecer servicios en la nube, entre ellas las que mas destacan a nivel mundial son [1]:

- **AWS**: Amazon Web Services [2] ofrece un gran contenido de servicios en la nube, tiene muchas similitudes a Google Cloud Platform ofreciendo herramientas muy parecidas a las usadas en este proyecto como EKS(Elastic Kubernetes Service), inicialmente se estudió la opción de desarrollar el proyecto en esta plataforma pero finalmente se optó por la nube de Google por las facilidades que da a nuevos usuarios.
- **Azure**: La nube de Microsoft es la más utilizada actualmente a nivel mundial ya que provee un uso muy profundo de las tres capas de la nube (*IaaS,SaaS y PaaS*).

3.2. CICD

Actualmente la herramienta más utilizada para aplicar la metodología CICD es Jenkins pero también se usan otras como **Gitlab CI** [3] la cual contiene un Registry para imágenes incorporado.

3.3. Google Kubernetes Engine

GKE es la plataforma de Kubernetes que nos ofrece Google en su nube, pero Amazon tiene **Elastic Kubernetes Service** la cual es muy parecida en funcionalidad pero sin una gestión automática de la herramienta tan alta.

3.4. Web de rol

En la actualidad hay páginas como `www.roll20.net` que ofrecen servicios parecidos a la aplicación que se va a desarrollar, esta web puede ayudar mucho a obtener ideas y contemplar situaciones que no se habían pensado. A diferencia de *maskdados.com* esta web está pensada como una plataforma para jugar online y no como una web de ayuda a las partidas de rol en mesa.

4 Metodología

Este proyecto ha sido creado haciendo uso de la metodología *Feature Driven Development* [4]. FDD es una metodología ágil que se basa en el desarrollo por características, la cual consiste en ir haciendo pequeñas iteraciones definidas en cinco procesos: desarrollar un modelo global, hacer un listado de características, planificar para cada función, diseñar y construir.
Estas iteraciones se centran en entregar una funcionalidad tangible del proyecto, ayudando a monitorearlo constantemente y comprobando que cada una de las funcionalidades haya sido creada correctamente.

Dentro de las fases de FDD, las tres primeras (Desarrollar un modelo global, hacer un listado de características y planificar cada función) son lineales y se hacen al principio del proyecto. Las últimas dos (Diseñar y construir) se hacen en cada iteración.

5 Requerimientos del sistema

Una vez se han tenido claros los objetivos del proyecto, se ha creado una tabla con los requerimientos que debe tener junto con su prioridad. [Tabla 1]

6 Planificación del trabajo

La planificación del trabajo cambió completamente cuando, a causa del COVID-19, se tuvo que hacer un reinicio del mismo proyecto. Inicialmente iba a ser un proyecto que se realizaría en la empresa MANGO y trataría sobre la implantación de un sistema de radiofrecuencia en el sistema de etiquetado dentro de las diferentes tiendas de la marca. A causa de la pandemia y de la finalización de mi estancia en la empresa, el proyecto estuvo un tiempo sin un rumbo claro mientras se estudiaban diferentes maneras de reestructurarlo. Inicialmente se planteó como un estudio de la metodología CICD aplicado en un pequeño caso y se dedicaron los meses de marzo y mayo a hacer una investigación de todo ello, centrándose en las diferentes fases de la integración/entrega/despliegue continuo.
En junio, cuando se decidió que el proyecto se entregaría en septiembre, se volvió a hacer otra reestructuración del proyecto hasta decidir el objetivo actual. Tras este proceso, y después de dividir el proyecto en sus cuatro pilares (GCP, CICD, GKE, API REST) se dedicaron los siguientes Sprints o iteraciones de una o dos semanas cada uno:

- **Sprint 0**: Configuración de la nube de Google Cloud Platform (GCP).

TABLA 1: REQUERIMIENTOS DEL SISTEMA

Requerimientos del sistema	
El sistema debe desarrollarse haciendo un uso completo de la nube de Google Cloud Platform (GCP).	alta
El sistema debe hacer uso de la metodología CICD utilizando Jenkins.	alta
La aplicación web del sistema debe desplegarse usando la herramienta de Google Kubernetes Engine (GKE).	alta
La aplicación web del sistema debe ser una API REST teniendo la parte del *back end* y *front end* completamente separadas.	alta
La parte *front end* de la web del sistema será desarrollada usando Angular.	alta
La parte *back end* de la web del sistema será desarrollada utilizando el framework de Sprint en Java 11.	alta
La aplicación web permitirá a los usuarios registrarse y hacer login en ella.	alta
La aplicación web permitirá a los usuarios crear fichas de personaje de juegos de rol a través de plantillas.	alta
La aplicación web permitirá a los usuarios consultar y editar sus fichas de personaje creadas.	alta
La comunicación entre el *back end* y el *front end* de la web del sistema se hará utilizando la seguridad del estándar JSON Web Token (JWT).	media
La aplicación web tendrá un dominio público y utilizará un certificado SSL.	media
La aplicación web ofrecerá a los usuarios la opción de crear plantillas personalizadas.	baja
La aplicación web debe ser responsive para todo tipo de tamaño de pantallas.	baja
La aplicación web debe ofrecer una experiencia de usuario fácil de utilizar.	baja

- **Sprint 1**: Instalación de Jenkins en GCP y su configuración inicial.
- **Sprint 2**: Implantación de la herramienta de Google Kubernetes Engine y creación del clúster donde se desplegará el *back end* y el *front end*.
- **Sprint 3**: Implementación de la conexión de Jenkins con el clúster de GKE usando *Services Acounts* de GCP.
- **Sprint 4**: Definición de la estructura de la aplicación web.
- **Sprint 5**: Diseño de los contenedores Docker de despliegue para el *front end* y el *back end*.
- **Sprint 6**: Diseño de la estructura de Kubernetes y codificación de sus manifiestos necesarios.
- **Sprint 7**: Diseño de la base de datos no relacional utilizando MongoDB Atlas.
- **Sprint 8**: Diseño de la fases de la metodología CICD y su implantación para el desarrollo de la aplicación web.
- **Sprint 9**: Implementación del *back end* de la aplicación web.
- **Sprint 10**: Implementación del *front end* de la aplicación web.
- **Sprint 11**: Implementación de la seguridad con JSON Web Token.
- **Sprint 12**: Implementación del dominio público *www.maskdados.com*.
- **Sprint 13**: Implementación del certificado SSL usando las herramientas que ofrece GKE.

En el Apéndice [A.1] se puede ver el diagrama de Gantt del proyecto.

7 ARQUITECTURA CLOUD

7.1. Google Cloud Platform

La base del proyecto consiste en realizar el sistema utilizando todas las herramientas posibles que ofrece la nube de Google Cloud.
Google Cloud Platform es la externalización de los servicios de computación que siempre ha ofrecido Google pero ahora ahora unificadas en una misma plataforma. Estos servicios incluyen aspectos como máquinas virtuales personalizables que son facturadas por uso, herramientas de gestión de contenedores, servicios de Big Data y plataformas de Machine Learning entre otros. Gracias a Google Cloud se podrá externalizar toda la infraestructura necesaria para el desarrollo de este proyecto, pudiendo así delegar a la nube todo lo referente al despliegue de la aplicación.
Google Cloud Platform(GCP) nos ofrece sus productos agrupados en diferentes partes según su función, algunos de esos grupos mas importantes son:

- **Computación**: Aquí están las diferentes herramientas que ofrece Google para la computación en la nube, desde la creación de máquinas virtuales con las características deseadas con Compute Engine, App Engine para crear aplicaciones escalables en diferentes lenguajes de programación o Google Kubernetes Engine la cual se utilizará en este proyecto.
- **Almacenamiento**: En este área están las diferentes herramientas de almacenamiento, tanto para tener discos duros en la nube como para la gestión de bases de datos.
- **Big Data**: En esta agrupación se encuentran diferentes aplicaciones que son utilizadas para trabajar con el Big Data, nos permiten analizar, procesar y visualizar datos de manera rápida.
- **Inteligencia Artificial**: En esta parte están todos los productos para trabajar con inteligencia artificial y hacer uso del Machine Learning.
- **Redes**: Los productos aquí presentes permiten controlar todo lo relacionado con la red: gestión de dominios, VPN, cortafuegos...

7.2. Componentes utilizados

Una vez estudiada y planteada la estructura que debería tener el proyecto, se decidió usar los siguientes componentes o servicios:

- **Máquinas virtuales**: En total se han creado un número de cuatro máquinas virtuales en Google Cloud en la región europe-west: tres de ellas son administradas completamente por Google Kubernetes Engine y la otra se ha creado usando la herramienta de Compute Engine donde se instalará Jenkins y se gestionará todo el CICD. Esta última ha sido nombrada **Jenkins** y establecida con unas características sencillas para correr el software de Jenkins: es una máquina del tipo e2-medium con 2 vCPU, 4GB de memoria RAM y con el sistema operativo Ubuntu 20.04.

- **Almacenamiento**: Se ha creado una unidad de almacenamiento de 200GB con la herramienta de **Storage** que es asignada a la VM de Jenkins.

- **Google Kubernetes Engine (GKE)**: Utilizando esta herramienta se han creado tres máquinas virtuales que serán los nodos de nuestro clúster de Kubernetes. En estos nodos será donde se desplegarán los pods de la aplicación web. Estas máquinas han sido construidas con 2 vCPU y 4GB de RAM, además tienen un disco duro persistente de 100GB cada una.

- **Container Registry**: Esta herramienta proporciona un almacenamiento privado y seguro de imágenes Docker dentro de Google Cloud Platform. En el proyecto se usa para la fase de entrega continua dentro del CICD ya que las imágenes Docker generadas se almacenarán allí.

- **Administración de Identidades y Accesos (IAM)**: IAM define quién tiene acceso a qué función de Google Cloud. Para ello, dentro de las diferentes opciones que ofrece este recurso, se hará uso de las **cuentas de servicio**: estas son cuentas con permisos definidos manualmente para una aplicación y no para un usuario, lo cual permitirá que los diferentes elementos de nuestra nube se puedan autenticar y comunicarse entre ellos de manera autónoma. En este proyecto se ha creado una cuenta de servicio con los permisos necesarios para acceder al clúster de Kubernetes, y ha sido asignada a la máquina virtual **Jenkins**.

- **Direcciones IP externas**: Para poder desplegar la aplicación públicamente se han requerido dos IP externas fijas globales que nunca cambien, ya que serán las direcciones que usará la aplicación web en el *back end* y el *front end*. Se han generado con esta herramienta dentro de la agrupación **Redes**, después serán asignadas a cada parte de la aplicación.

- **Cloud DNS**: Para poder hacer que la aplicación sea accesible al usuario se tenía que añadir un dominio para cada dirección IP externa. Este fue adquirido en los proveedores de dominio hostalia.com e ionos.es. Se adquirió *maskdados.com* y *maskdados.es*, dominios que serán usados para el *front end* y *back end* respectivamente. Una vez conseguidos, se utilizó la herramienta de Cloud DNS para crear una zona por cada dominio; configurándolas se asociará el nombre adquirido con la IP externa creada previamente.

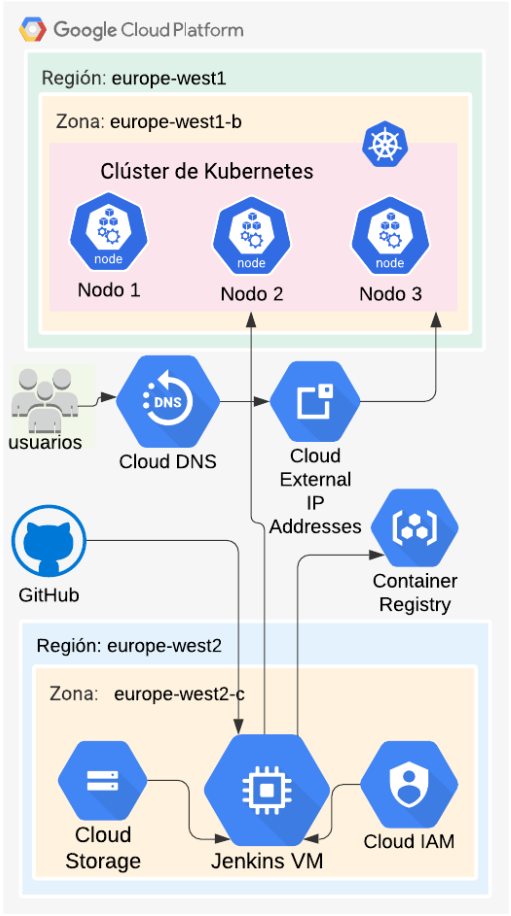

Fig. 1: Arquitectura en Google Cloud Platform

7.3. Costes asociados

Una de las ventajas que tiene Google Cloud Platform para nuevos usuarios es que ofrece 300$ gratis como crédito para usar sus servicios durante 90 días, esto ha servido para poder desarrollar el proyecto a un coste mínimo. *Gastos variables: Movimiento de GBs en Cloud Registry.

TABLA 2: COSTES ASOCIADOS POR MES

Costes asociados por mes	
1x Jenkins e2-Medium 2GB RAM	26.68€/mes
3x Nodos e2-Medium 4GB RAM	68.33€/mes
2x Cloud DNS zones	0.34€/mes
Gastos variables*	0,022€/mes x GB

8 KUBERNETES Y GKE

En esta sección se explicarán conceptos de contenedores y cómo se ha desarrollado el despliegue en Kubernetes

usando la herramienta de Google Kubernetes Engine y su estructura.

8.1. Containerización y Docker

Primero de todo se ha de comprender el concepto de la containerización, la cual permite desplegar aplicaciones auto contenidas. Esto quiere decir que se permite aislar nuestro código con únicamente las herramientas necesarias para ejecutarlo correctamente, lo cual hará que pueda usarse en cualquier máquina sin importar qué otro software tenga instalado; así se eliminan problemas de incompatibilidades y se asegura su ejecución en todo tipo de máquinas.
Cuando se habla de containerización se tiene que hablar de **Docker** [5], ya que es la tecnología mas usada para la creación de contenedores. Con él, podemos crear las imágenes de contenedores personalizadas que deseemos, subirlas a repositorios de imágenes públicos o privados o descargar imágenes creadas por otras personas para utilizarlas.
Las imágenes Docker se crean a partir de **Dockerfiles**, que son archivos de texto plano que contienen una serie de instrucciones para construir una imagen. Estas imágenes se construyen usando el comando de Docker *"build"* y luego se ejecutaran usando *"run"*.
La tecnología de Docker ha sido una de las bases para el desarrollo de este proyecto. Se han creado principalmente tres imágenes de contenedores:

- **Jenkins**: La primera imagen. Esta contiene todo lo necesario para poder configurar Jenkins y ejecutar cada una de las fases del CICD. Por ello, nuestro Dockerfile contiene todas las instrucciones para descargarse una imagen oficial de Jenkins que tenga preinstalado **Java 11** y una vez descargada instalar en ella diferente software. **Maven** para trabajar con nuestro proyecto de Java; **Docker** ya que este contenedor necesitará crear otros contenedores; **Kubectl** para poder hacer uso de Kubernetes; **gcloud** para poder acceder a nuestro clúster de **GKE**.
- **Back end**: Para desplegar la parte del *back end* de la aplicación web se creó una imagen a través de un **Dockerfile** que crearía un contenedor a partir de una imagen oficial de **Java 11**: expone el contenedor por el puerto 8095 que es donde **Spring Boot** está configurado para oír, transfiere el ejecutable **.jar** compilado previamente hacia nuestro contenedor y establece que al arrancar el contenedor se lance el ejecutable de Java para que inicie la aplicación.
- **Front end**: La imagen creada para la parte del *front end* ha sido construida con un **Dockerfile multistage**, que no es mas que generar un archivo Dockerfile que contiene más de una etapa. En este caso, en la primera etapa se crea una imagen a partir de una imagen oficial de **Node** que es necesaria para **Angular**, se hace un *"npm install"* para instalar las dependencias necesarias y se ejecuta un "npm run build" para construir la aplicación de Angular. En la segunda etapa se utiliza una imagen *"nginx:alpine"*, la cual es un **nginx** muy ligero, y se le añade toda la aplicación construida en la anterior etapa en la carpeta html de nginx. Esto hará que nuestra aplicación web esté disponible cuando accedamos al contenedor por su puerto 80.

8.2. Kubernetes y Google Kubernetes Engine

Kubernetes [6] es una plataforma open source considerada como un orquestador de contenedores, que automatiza las operaciones con ellos y ayuda a solucionar los problemas de los procesos manuales. Esta herramienta nos permite administrar un gran número de contenedores de manera fácil, ayuda a vigilar, organizar y controlar los contenedores. Como ejemplo, Kubernetes estará vigilando que el número de contenedores definidos estén activos; si uno de ellos se cae, volverá a levantar otro más hasta tener el número establecido.
Como este proyecto se está desarrollando en Google Cloud, se trabajará con Google Kubernetes Engine, el servicio de Kubernetes gestionado por Google que nos ofrece ventajas como una fácil creación de clústers, auto-escalabilidad, balanceos de carga, actualizaciones, reparaciones automáticas y un monitoreo de los logs. Un concepto importante que se ha de conocer es el de los manifiestos de Kubernetes. Estos son ficheros *.yaml* que contienen las instrucciones para crear los componentes de Kubernetes. En el Apéndice [A.6] se puede ver el manifiesto de una parte del *back end*.

8.3. Recursos en GKE

Para ello se usan diferentes recursos que ofrece Kubernetes en GKE, los cuales se describen a continuación:

- **Clústers**: Un clúster [7] es la base de Kubernetes, es el set de "nodos" que ejecutan aplicaciones en contenedores. Al ejecutar Kubernetes estás ejecutando un clúster.
- **Nodos**: Los nodos son cada máquina (ya sea física o virtualizada) que compone un clúster de Kubernetes. Cada nodo está gestionado por un componente máster y tiene los elementos necesarios para ejecutar los "pods".
- **Pods**: Es la unidad desplegable más pequeña que se puede crear y gestionar en Kubernetes. Es un conjunto de uno o más contenedores implantados en un único nodo. En este proyecto los pods son los contenedores Docker de nuestra aplicación, habrán pods del *back end* y del *front end*.
- **ReplicaSet**: Un ReplicaSet es un recurso que asegura que siempre se ejecute un número de réplicas de un pod determinado. Nos asegura que un conjunto de pods siempre está funcionando y disponible.
- **Deployment**: El Deployment es una unidad de alto nivel de Kubernetes que nos permite tener un control de nuestros ReplicaSet, controlar la escalabilidad, actualizar los pods, hacer despliegues automáticos y hacer rollbacks a versiones anteriores. Un ReplicaSet únicamente se encarga de tener activos un número de pods, pero solamente el Deployment puede actualizar las imágenes que usan esos pods. Cuando se crea un Deployment automáticamente se crea un ReplicaSet asociado.
- **Service**: Un Service es una abstracción que define un conjunto de pods que implementan un único microservicio. Por ejemplo, en este proyecto todos los pods del

back end tienen un servicio único. Hay varios tipos de servicios. En este proyecto se han utilizado los de tipo **NodePort**, el cual nos crea una IP interna dentro del clúster y nos expone un puerto entre el rango 30000-40000 al exterior desde donde será accesible.

- **Ingress**: GKE nos permite crear un objeto Ingress para el balanceo de cargas de HTTP(S). Este proporciona las reglas para enrutar el tráfico externo HTTP(S) a las aplicaciones que se ejecutan en el clúster. El recurso Ingress se asocia con uno o más recursos Service y también puede ser asociado a IPs estáticas proporcionadas por Google y "ManagedCertificates".

- **ManagedCertificate**: Google ofrece certificados SSL administrados por él, estos se aprovisionan, renuevan y administran para los nombres de dominio.

8.4. Estructura utilizada en GKE

Al definir la estructura del proyecto se decidieron dos partes claras que serían desplegadas en GKE, el *back end* y el *front end*. Se decidió que ambos compartieran clúster con los mismos nodos, pero cada uno con su propia estructura de Kubernetes. A continuación se mostrará una imagen de la estructura del *front end*.

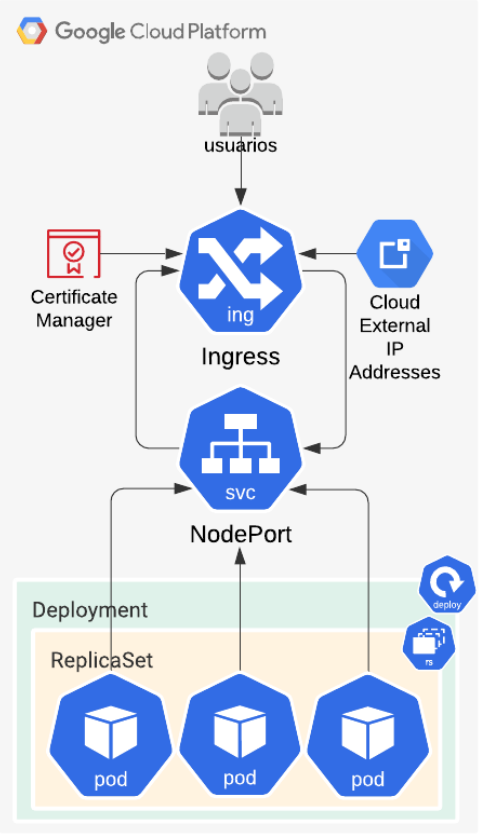

Fig. 2: Estructura Kubernetes *front end*.

Como se puede observar, la estructura ha sido diseñada con tres Pods administrados que ejecutan la aplicación web. Estos son creados en los nodos que tiene nuestro clúster. Tienen un NodePort que expone un puerto entre el 30000 y 40000 al exterior y un Ingress asociado que contiene un ManagedCertificate para añadir el certificado SSL y ser accesible vía HTTPS. Se le establece al Ingress una IP externa estática creada previamente en Google Cloud. Esta IP tiene asociada el nombre de domino *maskdados.com*. Por último, los usuarios accederán a nuestra aplicación a través del Ingress, el cual hará de balanceador de carga entre los diferentes pods y nodos.

8.5. Escalabilidad en GKE

Una de las grandes ventajas que tiene usar Google Kubernetes Engine es la gran escalabilidad que ofrece. Simplemente con un comando se puede modificar el número de Nodos utilizados en el clúster para ajustar a la demanda necesaria, indicarle cuántos Pods se quiere tener activos, cuánta RAM dedicarle a cada Pod y muchas más opciones. Por otro lado, GKE nos ofrece cuatro vías de autoescalado gestionadas por la misma nube:

- **Carga de trabajo- horizontal**: Este autoescalado se basa en añadir o eliminar Pods según la utilización del CPU u otras métricas personalizadas.

- **Carga de trabajo- vertical**: El autoescalado de carga de trabajo vertical analiza continuamente el uso de la CPU y la memoria de los pods y modifica los pods para encajar en ello.

- **Infraestructura - horizontal**: Este autoescalado se basa en añadir o eliminar nodos según la utilización de los mismos.

- **Infraestructura - vertical**: El autoescalado a nivel de infraestructura vertical modifica los nodos para las necesidades de los pods utilizados.

9 CICD

CICD [8] es una metodología para distribuir aplicaciones con frecuencia, mediante el uso de la automatización en las etapas del desarrollo de las aplicaciones. Encarna una cultura, principios y prácticas que permiten a los desarrolladores tramitar cambios de código de manera más frecuente y fiable.
Las siglas CICD hacen referencia a los aspectos de "Integración Continua (CI)", "Entrega Continua (CD)" y "Despliegue Continuo (CD)", los cuales han sido integrados en el proceso de desarrollo de la aplicación web.
A continuación se describirá cada uno de ellos y su aplicación en este proyecto.

9.1. Integración Continua

La Integración Continua ayuda a implementar código con mayor frecuencia y añadir pequeños cambios y modificaciones a la aplicación, todo esto siguiendo un proceso automatizado que implemente ese nuevo código. De esta manera se evita tener que hacer toda la fusión del trabajo de todos los desarrolladores en un único día y continuamente se va integrando nuevo código. Una vez se fusionan los cambios se validan a través de una serie de pruebas automatizadas definidas en esta fase de la Integración Continua. Si no hay ningún error el código nuevo quedará integrado.

9.2. Entrega continua

Una vez la nueva integración de código ha sido implementada correctamente y se ha comprobado que no hay ningún error se pasa a la fase de Entrega Continua, la cual automatiza la distribución del código hacia un repositorio. Esto nos permite tener un repositorio con nuestra última versión del código lista para ser desplegada.

9.3. Despliegue continuo

Esta última fase de CICD se encarga de automatizar el despliegue de una aplicación, que va desde la última versión valida del código que está en el repositorio donde se ha entregado en la fase anterior, hasta la plataforma de despliegue.

9.4. Jenkins

Una de las herramientas mas utilizadas cuando se implementa CICD es Jenkins, el cual es un sistema desplegado en un servidor que nos permite automatizar procesos. Esto hace que sea una herramienta ideal para implantar las fases de integración, entrega y despliegue continuo.
Algunos de los conceptos importantes de Jenkins que se han de tener claros en este proyecto son:

- **Plugins**: Los plugins ofrecen una gran cantidad de funcionalidades extras que se pueden añadir al servidor de Jenkins. Estos han sido creados por diferentes desarrolladores para solucionar algún problema existente o para integrar alguna tercera tecnología al servidor.
- **Jobs**: Los Jobs son las tareas configuradas en Jenkins. Contienen una serie de instrucciones a ejecutar.
- **Pipelines**: Uno de los conceptos más importantes y utilizados en este proyecto son las pipelines, estas definen el flujo de trabajo por donde tiene que pasar el código para llegar a producción. Este flujo viene definido por los "Stage" que son las diferentes fases que tiene el ciclo de vida de la Pipeline, y los "Steps", que son las tareas que tiene esa fase. En CICD, la Pipeline tendrá todas las fases necesarias que van desde que el desarrollador añade código a un repositorio, hasta que se despliega la nueva versión de la aplicación.
- **Jenkinsfile**: Jenkinsfile [9] es un fichero de texto el cual contiene todas las instrucciones para crear la Pipeline con sus fases. Este fichero se almacena y se versiona junto con el código de la aplicación en el repositorio, define cada "Stage" y cada "Step" de nuestra Pipeline de CICD. Gracias a tenerlo en el repositorio obtenemos todas las ventajas del control de versiones y nuestras fases de integración, entrega y despliegue continuo pueden variar según las necesidades del proyecto. En el Apéndice [A.5] se muestra el código del archivo Jenkinsfile del *back end*.

9.5. Estructura CICD

Cuando se decidió la estructura que tendría el proyecto y se creó la arquitectura de Google Kubernetes Engine necesaria, se comenzó a trabajar con la creación de dos Pipelines que implementarían toda la metodología CICD para el desarrollo del *back end* y el *front end* de la aplicación web.
Una vez instalada y configurada nuestra máquina virtual de Jenkins en Google Cloud Platform se añadió una cuenta de servicio (IAM) de GCP para poder comunicar Jenkins con el clúster de Kubernetes y el Registry de imágenes en la nube de Google.
Para implementar la automatización del CICD se configuró el repositorio de código de *Github* para que lanzara un *"Webhook"* cada vez que se hiciese un *commit* a nuestro repositorio. Esto lanzaría un *Trigger* a la IP de nuestra máquina Jenkins que arrancaría la Pipeline con el nuevo código.
Esta Pipeline se definió con las siguientes fases:

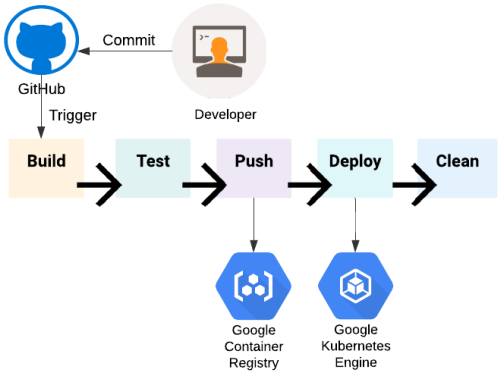

Fig. 3: Pipeline CICD.

- **Build**: Este primer "Stage" se encarga de hacer un *build* de nuestro código, en Java se usará Maven para ello y en Angular crearemos la imagen a través del Dockerfile, ya que esta tiene el *build* incorporado.
- **Test**: Esta fase solo está presente en el pipeline del *back end*. Aquí se utiliza Maven para ejecutar los test unitarios de nuestra aplicación *Spring Boot*. Esto ejecuta los test nuevos y los test presentes previamente. Si hay algún fallo en ellos, el proceso de automatización terminará y nos indicará donde está el error. Hasta este punto consiste la parte de la **Integración Continua**.
- **Push**: La fase de *Push* implementa la parte de **Entrega Continua** y crea la imagen a partir del Dockerfile definido, a continuación esta se guarda en *Google Container Registry*.
- **Deploy**: Deploy implementa la última parte del CICD, el **Despliegue Continuo**. Aquí desplegamos la aplicación en *Google Kubernetes Engine* haciendo uso de los *manifiestos* de Kubernetes creados previamente y el comando *kubectl*, que como se tienen las credenciales de el clúster gracias a la cuenta de servicio *IAM*, se podrá desplegar desde la máquina virtual de Jenkins.
- **Clean**: Esta última fase se ha creado por la necesidad surgida de limpiar Google Container Registry de imágenes previas guardadas. Esto se ha decidido para minimizar los costes que GCP tiene por almacenar datos en su Registry. Borramos todas las imágenes guardadas en el registro que no tengan el *tag:latest*, y eliminamos las imágenes generadas en la máquina virtual

de Jenkins, dejando únicamente la última versión correcta guardada.

10 MASKDADOS.COM

MASKDADOS es una aplicación web de ayuda a los juegos de rol que está pensada como herramienta para la gestión de partidas en mesa.
Para entender la utilidad de esta aplicación se hará una breve explicación de los siguientes conceptos:

- **Juego de Rol**: Es un juego interpretativo en el que uno o más jugadores desempeñan un papel o rol a lo largo de una historia o trama. Hay dos tipos de jugadores, el director de juego y los personajes.
- **Director de juego**: El curso de las partidas está supervisado por el director de juego, que es quien cumple las funciones de narrador de la historia y de mediador entre jugadores.
- **Personaje**: Los jugadores no narradores interpretarán a diferentes personajes. Estos suelen ser personajes imaginarios que siguen unas pautas de juego preestablecidas conocidas como sistema de juego, los datos de esos personajes estarán escritos en fichas.
- **Fichas**: Las fichas contienen todos los datos de los personajes que juegan en la historia. Según el sistema de juego pueden tener datos como características físicas o habilidades. Cuando se juega, la ficha es necesaria para poder desempeñar los papeles y acciones que hacen los personajes.

Teniendo estos conceptos claros, se ha creado la aplicación web para ofrecer ayudas a la partida como un lugar online donde gestionar todos los datos de las fichas de los personajes y tener todo lo necesario para poder jugar a los juegos rol. De esta manera se soluciona uno de los problemas que más suele ocurrir, el no tener las fichas de los personajes cuando se va a jugar, ya que ahora siempre tendremos un lugar online y no se necesitará el papel físico.

10.1. Aplicación web

Uno de los primeros objetivos fue crear la web separando claramente la parte lógica y de gestión de la base de datos en el *back end* y la parte visual con la que interactúa el usuario como el *front end*.
Para ello se han usado dos tecnologías muy diferentes que interactúan entre ellas a partir de peticiones web, la parte front hará peticiones HTTP al *back end*, el cual le devolverá los datos que requiere.

10.1.1. Spring Boot

Spring [10] es un framework de código abierto que facilita el desarrollo de aplicaciones web en Java. Tiene una estructura modular y una gran flexibilidad para implementar diferentes tipos de arquitecturas.
En este proyecto se ha usado Spring Boot, la cual es una herramienta que simplifica aún más el desarrollo de las aplicaciones web creadas con el framework de Spring y contiene un *Tomcat* embebido que se lanza al ejecutar nuestra aplicación, el cual nos servirá como servidor para escuchar las peticiones HTTP.
Para la construcción de la parte *back end* se ha decidido usar el IDE *Eclipse*, el cual tiene un plugin para Spring que facilita su programación. En cuanto a la estructura del código, se ha trabajado en un proyecto Maven de Java, que ofrece un archivo de configuración *POM* que contiene todas las dependencias necesarias.
Se ha decidido hacer una aplicación basada en micro servicios multi modular dentro del mismo proyecto Java. En la aplicación se han generado diferentes *Endpoints*, los cuales reciben peticiones HTTP, y si estas están autenticadas correctamente devuelve el valor solicitado. La aplicación contiene los siguientes micro servicios:

- **AuthController**: Este micro servicio es muy importante ya que trata con todo lo referente a la seguridad de la web, tramita los registros de usuarios y se encarga de generar los tokens de autenticación a los que han hecho login correctamente.
- **FichaController**: FichaController contiene los *Endpoints* que tratan con las fichas de personajes que contiene la base de datos, como por ejemplo obtener el listado de personajes de un usuario, editar la ficha de un personaje y obtener sus datos. Para poder acceder a este servicio se ha de estar autenticado con un Token válido.
- **PlantillaController**: Cuando un usuario decide crear una ficha lo hace través de una plantilla, la cual contiene la información para que el *front end* sepa como mostrarla. En este servicio están los "Endpoints" para obtener las plantillas de la base de datos. Se ha de estar autenticado con un Token válido.

Para probar el código se han hecho pruebas unitarias a los servicios de Spring con Junit 5 y se ha utilizado la herramienta PostMan [11] para probar que los *Endpoints* funcionen correctamente.En el Apéndice [A.8] se puede ver una prueba con Postman al *endpoint* de login en AuthController.

10.1.2. Angular

Para crear la parte *front end* se ha decidido utilizar el framework de Angular [12], el cual es un framework opensource desarrollado por Google que facilita la programación de aplicaciones web de una sola página (Single Page Application). Gracias a ello la página no tiene que recargar cuando se navegue por ella ya que todo estará previamente cargado.
La estructura de Angular está definida por componentes los cuales cada uno contiene tres partes, los estilos *CSS*, la estructura de la vista HTML y un archivo *.ts* escrito en typescript que contiene toda la lógica del componente. Además una de las ventajas de Angular es que disponemos de servicios que son utilizados para tramitar datos entre componentes o para hacer peticiones a *Endpoints* como nuestro *back end* en Spring Boot.
Dentro de la aplicación se han creado componentes para cada elemento de la web. Los mas importantes son:

- **LoginComponent**: Este componente contiene todo lo necesario para mostrar la pantalla de login y poder validar los campos escritos. Cuando el usuario hace clic

a login este elemento usará un servicio que hace una petición POST a la API REST de Login del micro servicio AuthController. Apéndice [A.4.3]

- **RegisterComponent**: RegisterComponent contiene el formulario de registro donde los campos son validados antes de enviar al micro servicio de registro, el cual guardará el usuario en la base de datos. Apéndice [A.4.2]
- **Plantilla-dungeon-hack-Component**: Una de las plantillas creadas ha sido la de el juego de rol *Dungeon Hack*, cuando el usuario quiere crear una ficha de este juego, el componente lee el archivo JSON de la plantilla obtenido de la BD y muestra una ficha de ese juego para rellenar, validando cada campo escrito. Apéndice [A.4.4]
- **Ficha-dungeon-Component**: Este component es muy similar al de Plantilla-dungeon-hack-Component con la diferencia que recibirá de la base de datos todos los datos de la ficha, los cuales mostrará de manera similar a la ficha real del juego. El usuario podrá editar los campos de la ficha que quiera y guardar esos cambios en la base de datos. Apéndice [A.4.5].
- **Card-ficha-component**: Cuando un usuario selecciona "Mis fichas" la aplicación Angular hará una consulta al *back end* y cargará en pantalla unas tarjetas con información mínima de las fichas de los diferentes personajes del usuario. Si el usuario selecciona una de ellas, se hará otra llamada al *back end* para obtener la información restante y llamará al componente de la ficha respectiva. Apéndice [A.4.1].

En el Apéndice [A.2] se dispone de un diagrama de secuencia para la creación de una ficha.

10.1.3. Seguridad

La aplicación web tiene dos puntos importantes en cuanto a la seguridad, uno de ellos es la implantación de certificados SSL para poder acceder vía HTTPS y que la conexión sea segura. La otra parte es la referente a la comunicación entre Spring Boot y Angular, ya que como el *back end* tiene una serie de "Endpoints" públicos cualquiera podría acceder y obtener información de la base de datos o del sistema. Para evitarlo se ha implantado la seguridad con JSON Web Token (JWT), el cual es un estándar de código abierto basado en JSON para crear tokens de acceso que nos permiten segurizar las comunicaciones entre un cliente y un servidor. Este token se genera a partir de un secreto. Cuando el usuario hace un login correctamente, el *back end* devuelve a la petición de login un token que el navegador del cliente guarda en su ordenador. Este token tiene una fecha de caducidad y hasta entonces las futuras peticiones HTTP que se hagan desde Angular tienen incorporado en el header el token generado. Spring Security es un plugin de Spring boot el cual está añadido al POM del proyecto, gracias a él podemos configurar los recursos CORS para definir quién accede a que servicio REST, ya sea requiriendo un rol especial o simplemente comprobando si tiene un token válido. En la aplicación web *maskdados.com*, todos los servicios REST están configurados para requerir un token válido excepto los de iniciar sesión y registrarse. Apéndice [A.7]

10.1.4. MongoDB

Después de definir la estructura de la aplicación web se hizo un estudio para decidir qué tipo de base de datos se implementaría, y al final se optó porque sería una base de datos no relacional. Esto fue así por el tipo de estructura que se trabajaría en la web ya que se guardarían muchos tipos de fichas de personaje con campos y tamaños muy diferentes los cuales requerirían un trabajo mucho mayor en tablas relacionales, por eso se decidió, por la libertad que ofrece en este aspecto MongoDB. En cuanto al lugar de despliegue de la base de datos se optó por varias opciones. Una de ellas era desplegar un Docker en una máquina virtual de GCP con MongoDB en ella, pero se descartó por el alto coste económico que tendría añadir otra máquina. Por eso al final se decidió usar la plataforma de *MongoDB Atlas*, ya que ofrece de manera gratuita hasta 512MB de almacenamiento usando un vCPU compartido, lo cual era más que suficiente para los requerimientos actuales.

11 RESULTADOS

En este punto se expondrán los resultados obtenidos teniendo en cuenta los objetivos establecidos en el segundo punto de este informe y los requerimientos del sistema.

11.1. Google Cloud Platform

El objetivo era montar la arquitectura en GCP y se ha cumplido de manera correcta, todo el despliegue de la aplicación y el servidor de Jenkins está montado en la nube de Google. Se hacen uso de muchos elementos de ella excepto para la base de datos por los motivos explicados en el punto anterior.

11.2. CICD

El segundo objetivo era desarrollar la aplicación haciendo uso de la metodología CICD, y ha sido otro objetivo cumplido; toda la aplicación ha sido desarrollada teniendo la Pipeline de CICD funcionando. Cuando esta fue configurada correctamente, cada vez que se hacía un commit del código al repositorio de *Github*, todos esos cambios se introducían en el flujo CICD hasta llegar al despliegue final, automatizando así todo el despliegue final de la aplicación y con ello poder centrarse únicamente en el desarrollo de la aplicación web.

11.3. Google Kubernetes Engine

Este objetivo requería desplegar toda la aplicación haciendo uso de la ha herramienta de Kubernetes Engine que ofrece Google y ha sido cumplido en su totalidad. Las dos partes de la aplicación web (Spring Boot y Angular) están corriendo en el clúster de GKE dónde están replicadas tres veces, y el cual contiene tres nodos donde desplegar la aplicación. Además se puede escalar la aplicación vertical u horizontalmente con un simple comando según la necesidad.

11.4. MASKDADOS.COM

La aplicación web maskdados.com ha sido desarrollada aunque no se han podido cumplir todos los requerimientos

establecidos por falta de tiempo. Aun así se ha conseguido un muy buen progreso y una muy buena base donde, con toda la estructura del proyecto montada, solo hace falta dedicarle horas en programación web para añadir más funcionalidades.

11.4.1. Requerimientos cumplidos

La web es funcional y es accesible vía HTTPS. Los datos son seguros y el usuario se conecta directamente a la web en Angular, la cual hace peticiones a la API REST de Spring Boot. Los usuarios pueden registrarse en la web y sus datos quedan guardados en la Base de datos. Las contraseñas son cifradas en el *back end* con un encoder antes de guardarse en MongoDB y las conexiones entre Angular y Spring Boot también son cifradas gracias al certificado SSL que se ha añadido al Endpoint de Spring Boot. Los usuarios pueden crear fichas del juego de rol *Dungeon Hack* y luego consultaras o editarlas. En el Apéndice [A.2] se puede ver el diagrama de casos de uso.
Por último , se puede acceder a la web a través del siguiente dominio:
https://www.maskdados.com

Fig. 4: Página inicial +kDa2

11.4.2. Requerimientos no cumplidos

Por falta de tiempo no se han podido cumplir los tres últimos requerimientos de prioridad baja. Ahora mismo la web solo permite crear fichas a través de plantillas pero no crear la propia plantilla y, aunque la web es responsive en aproximadamente un 50 %, hay partes que no están adaptadas para pantallas pequeñas.

12 Conclusiones

Como se ha visto en el informe, se han cumplido casi en su totalidad todos los objetivos establecidos. Esto ha sido posible gracias a la implantación metódica de cada una de las partes.
Como conclusión global de proyecto, se ha de decir que la elaboración del mismo ha sido un continuo desafío y aprendizaje desde el día uno. Los conocimientos adquiridos en el momento de finalizar el proyecto difieren enormemente de los iniciales.
Por eso mismo, el resultado final es muy satisfactorio y, aunque el trabajo realizado ha sido muy duro hasta llegar al punto actual, ha valido la pena.

12.1. Problemas encontrados

A la hora de realizar el trabajo se han encontrado algunos problemas. El primero de todos era la falta de experiencia con las tecnologías tratadas; previamente al proyecto solo se había trabajado con Spring Boot y todas las demás tecnologías han requerido de investigación y aprendizaje profundo para poder desarrollar este proyecto.
Otro problema encontrado ha sido el no gestionar correctamente el almacenamiento de las máquinas virtuales usadas, pues por ello se tuvo que reiniciar la parte de Jenkins desde 0 ya que la máquina anterior quedó inhabilitada con todo el trabajo hecho.

12.2. Futuras líneas de mejora

Una vez construido todo lo descrito en este informe, solo falta dedicarle horas de trabajo a la aplicación web.
Como futuros objetivos, se añadirían más plantillas de diferentes juegos rol, se podrían crear salas donde los jugadores puedan unirse con sus fichas y usarlas para poder hacer acciones como hablar en un chat o lanzar dados.
Otra línea de mejora futura sería la implantación de una IA que, a partir de una foto a una ficha física, pueda crearla en la aplicación web.
Y por último, se podría crear una aplicación nativa para Android o IOS que utilice la API REST de Spring Boot para poder ser más accesible para usuarios móviles.

Agradecimientos

Agradecer al tutor de mi proyecto, Fernando Vilariño, su ayuda y consejos para la orientación de este trabajo a lo largo de sus meses de elaboración. Agradecer a mi familia por todo el soporte incondicional durante mi etapa universitaria.

Referencias

[1] Forbes, https://www.forbes.com/sites/bobevans1/2017/06/05/meet-the-cloud-wars-top-10-the-worlds-most-powerful-cloud-computing-vendors/1b86f0a34e1e [En linea]
[2] Aws, https://aws.amazon.com/es/what-is-aws/ [En linea]
[3] GitLab CI, https://about.gitlab.com/stages-devops-lifecycle/continuous-integration/ [En linea]
[4] FDD, http://www.agilemodeling.com/essays/fdd.htm
[5] Turnbull, James (2014) The Docker Book.
[6] Redhat, https://www.redhat.com/es/topics/containers/what-is-kubernetes [En linea]
[7] RedHat, https://www.redhat.com/es/topics/containers/what-is-a-kubernetes-clúster [En linea]
[8] Paul M. Duvall (2007), Continuous Integration improving software Quality and reducing Risk.
[9] Jenkins, https://www.jenkins.io/doc/book/pipeline/jenkinsfile/ [En linea]
[10] Spring, https://spring.io/projects/spring-framework [En linea]
[11] PostMan, https://learning.postman.com/docs/getting-started/introduction/ [En linea]
[12] Angular, https://angular.io/docs [En linea]

APÉNDICE

A.1. Diagrama de Gantt

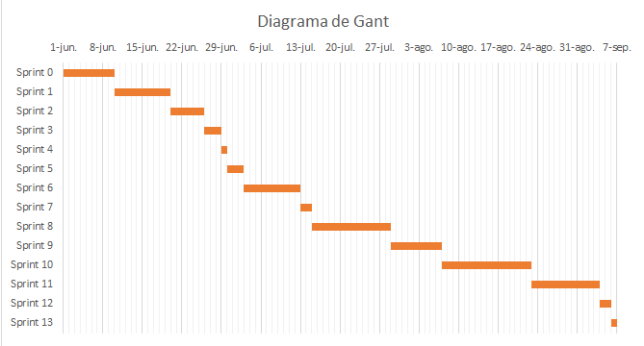

Fig. 5: Diagrama de Gantt

A.2. Diagrama de Secuencia

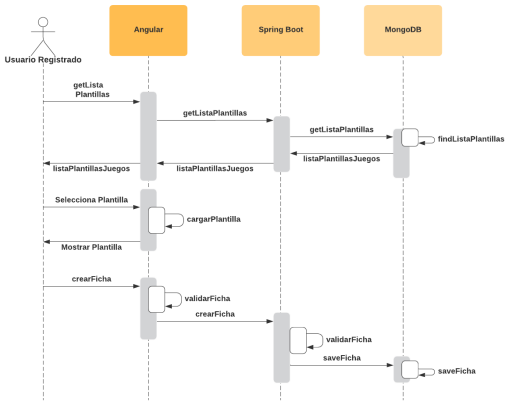

Fig. 6: Creación de ficha por plantilla

A.3. Diagrama de casos de uso

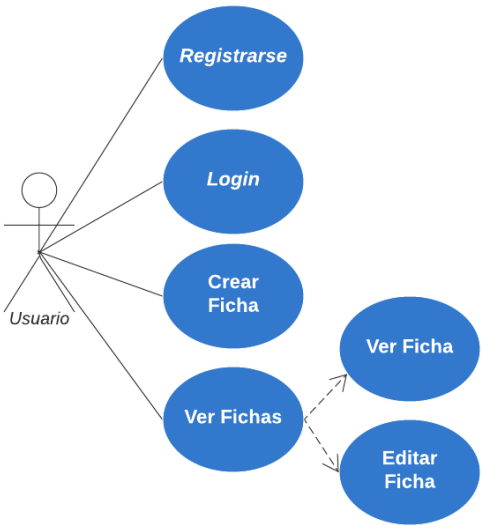

Fig. 7: Diagrama casos de uso

A.4. MASKDADOS.COM

A.4.1. Lista Fichas

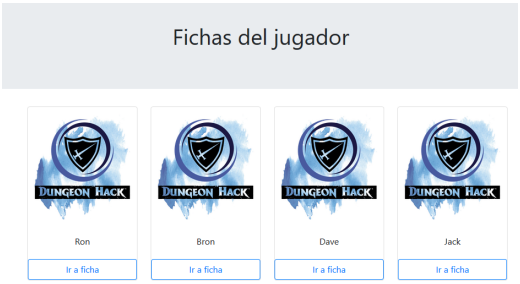

Fig. 8: Sección Mis fichas.

A.4.2. Registro

Fig. 9: Registro inválido.

A.4.3. Login

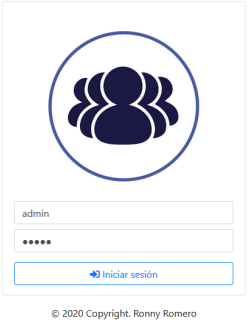

Fig. 10: Login

A.4.4. Crear ficha

Fig. 11: Crear ficha Dungeon Hack.

A.4.5. Editar Ficha

Fig. 12: Consultar y editar ficha.

A.5. Jenkinsfile Spring Boot

```
pipeline {
  environment {
    PROJECT = "Projectok8s"
    APP_NAME = "rolApp"
    PROJECT_ID = "projectok8s-286915"
    SOURCE_IMAGE = "springback"
    HOSTNAME = "eu.gcr.io"
    IMAGE_REP = "springback"
  }

    agent any
        stages {
            stage('Build') {
                steps {
                    sh "chmod +x mvnw"
                    sh "rm -rf ~/.m2/repository "
                    sh './mvnw -B -DskipTests clean install -Dhttps.protocols=TLSv1.2 '
                }
            }
            stage('Test') {
                steps {
                    echo 'test desde jenkins'
                    sh "./mvnw test"
                }
            }
            stage('Push') {
                steps {
                    sh "docker build -t ${HOSTNAME}/${PROJECT_ID}/${IMAGE_REP} ."
                    echo 'PUSH'
                    sh "whoami"
                    sh "docker push ${HOSTNAME}/${PROJECT_ID}/${IMAGE_REP}"
                }
            }
            stage('Deploy') {
                steps {
                    echo 'DEPLOY'
                    sh "kubectl apply -f manifest.yaml"
                    sh 'kubectl rollout restart deployment.apps/spring'
                    sh "kubectl apply -f ingresAndSslSpring.yaml"

                     }
                }
            stage('CleanRepository') {
                steps {
                    echo 'Borramos la imágen local creada y las imagenes del repositorio antiguas'
                    sh "docker rmi -f ${HOSTNAME}/${PROJECT_ID}/${IMAGE_REP}"
                    sh "gcloud container images list-tags eu.gcr.io/projectok8s-286915/springback
                     --filter='-tags:*' --format='get(digest)' --limit=unlimited | xargs -I {arg}
                      gcloud container images delete  \"eu.gcr.io/projectok8s-286915/springback@{arg}\" --quiet"
                    }
                }
        }
}
```

Fig. 13: Código de la *pipeline* de Spring Boot.

A.6. Manifiesto Kubernetes

```
pipeline {
  environment {
    PROJECT = "Projectok8s"
    APP_NAME = "rolApp"
    PROJECT_ID = "projectok8s-286915"
    SOURCE_IMAGE = "springback"
    HOSTNAME = "eu.gcr.io"
    IMAGE_REP = "springback"
  }

    agent any
        stages {
            stage('Build') {
                steps {
                    sh "chmod +x mvnw"
                    sh "rm -rf ~/.m2/repository "
                    sh './mvnw -B -DskipTests clean install -Dhttps.protocols=TLSv1.2 '
                }
            }
            stage('Test') {
                steps {
                    echo 'test desde jenkins'
                    sh "./mvnw test"
                }
            }
            stage('Push') {
                steps {
                    sh "docker build -t ${HOSTNAME}/${PROJECT_ID}/${IMAGE_REP} ."
                    echo 'PUSH'
                    sh "whoami"
                    sh "docker push ${HOSTNAME}/${PROJECT_ID}/${IMAGE_REP}"
                }
            }
            stage('Deploy') {
                steps {
                    echo 'DEPLOY'
                    sh "kubectl apply -f manifest.yaml"
                    sh 'kubectl rollout restart deployment.apps/spring'
                    sh "kubectl apply -f ingresAndSslSpring.yaml"

                     }
                }
            stage('CleanRepository') {
                steps {
                    echo 'Borramos la imágen local creada y las imagenes del repositorio antiguas'
                    sh "docker rmi -f ${HOSTNAME}/${PROJECT_ID}/${IMAGE_REP}"
                    sh "gcloud container images list-tags eu.gcr.io/projectok8s-286915/springback
                     --filter='-tags:*' --format='get(digest)' --limit=unlimited | xargs -I {arg}
                      gcloud container images delete  \"eu.gcr.io/projectok8s-286915/springback@{arg}\" --quiet"
                    }
                }
        }
}
```

Fig. 14: Archivo .yaml del manifiesto del *back end*.

A.7. Jason Web Token

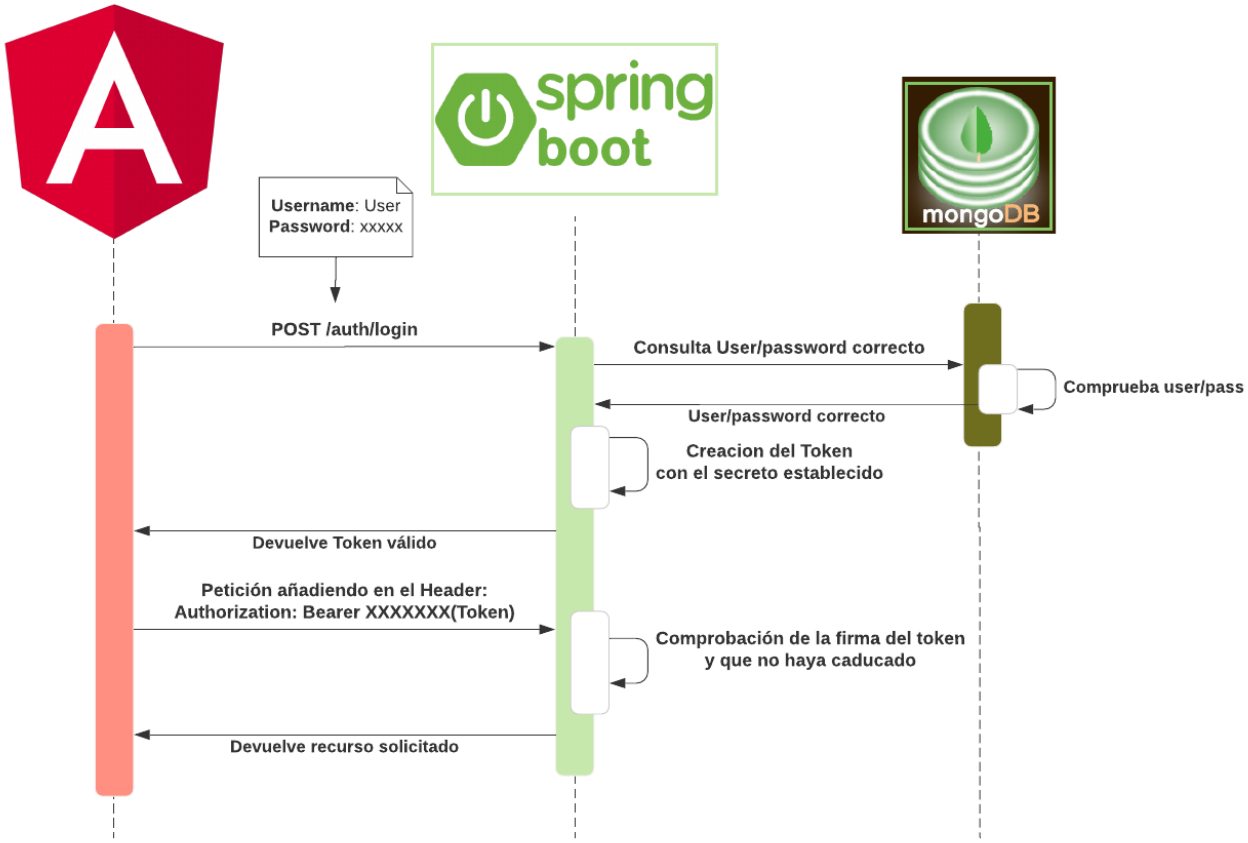

Fig. 15: Diagrama JWT

A.8. Postman

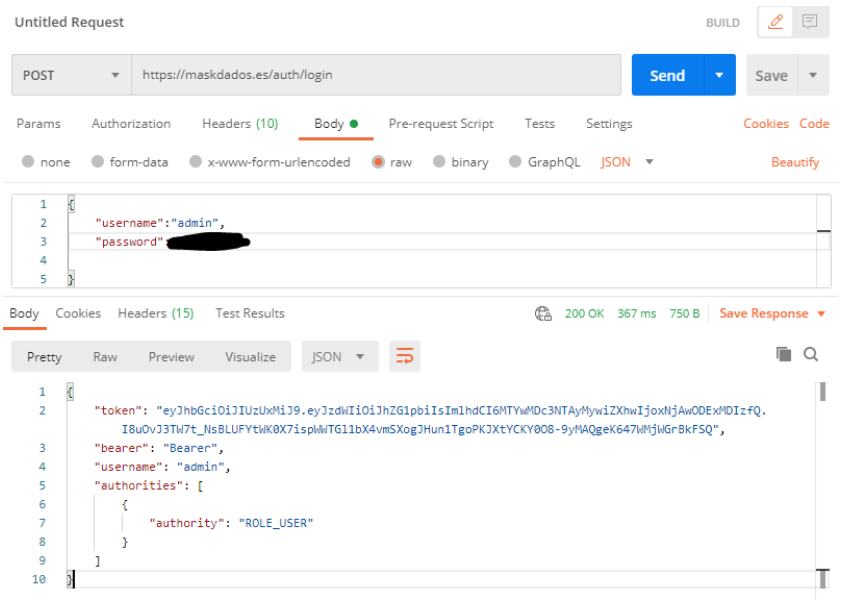

Fig. 16: Prueba Login Postman