UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA EN INFORMÁTICA

Gestor de jobs en entorno Cloud

Mohamed Essalhi Ahamyan

Julio, 2024

GESTOR DE JOBS EN ENTORNO CLOUD

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Tecnologías y Sistemas de Información

GRADO EN INGENIERÍA INFORMÁTICA

COMPUTACIÓN

Gestor de jobs en entorno Cloud

Autor: Mohamed Essalhi Ahamyan

Tutor académico: David Vallejo Fernández

Tutor de empresa: Pedro María Castellanos Sánchez-Carnerero

Julio, 2024

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL	SECRETARIO
Fdo.:	Fdo.:	Fdo.:

Abstract

Currently, task automation is a growing necessity. From scheduled nightly processes to tasks triggered by specific events, these functionalities are becoming increasingly common. This project addresses the development of an application for intelligent scheduling of periodic tasks using cloud computing environments.

This project is developed in collaboration with NTT Data, a global leader in information technology services, as part of the FORTalecimiento de la Empleabilidad (FORTE) program at the Escuela Superior de Informática (ESI) in Ciudad Real. This program provides students the opportunity to integrate into business projects while completing their Trabajo Fin de Estudios (TFE). The task scheduling application detailed in this document will be evolved and used by NTT Data for commercial purposes.

The application is based on a microservices architecture. This architecture allows the efficient integration of various technologies specifically adapted for each of the proposed objectives. This modular structure optimizes the flexibility and scalability of the system, thus supporting the load of different users. It also facilitates independent maintenance of each component.

The microservices network includes a central component for background task scheduling using cloud computing services. Additionally, extra microservices are integrated for user management and visualization of logs generated by tasks executed in the cloud.

To improve the user experience, an intuitive and user-friendly interface has been designed to significantly simplify interaction with the application, making task scheduling more accessible and efficient.

Furthermore, Inteligencia Artificial (IA) techniques are integrated to assist users in scheduling tasks. This includes the development of an intelligent assistant that will enable users to define schedules efficiently and adaptively in the cloud environment.

As the main result of the project, a web application has been developed, potentially allowing NTT Data's clients to benefit from task scheduling, freeing them from technical complexities and allowing them to focus on defining their tasks.

Resumen

En la actualidad, la automatización de tareas programadas es una necesidad creciente. Desde procesos nocturnos programados hasta tareas desencadenadas por eventos específicos, estas funcionalidades son cada vez más comunes. Este proyecto aborda el desarrollo de una aplicación para la programación inteligente de tareas periódicas, utilizando entornos de computación en la nube.

Este proyecto se desarrolla en colaboración con NTT Data, líder global en servicios de tecnología de la información, como parte del programa FORTalecimiento de la Empleabilidad (FORTE) de la Escuela Superior de Informática (ESI) de Ciudad Real. Este programa proporciona a los estudiantes la oportunidad de integrarse en proyectos empresariales mientras completan su Trabajo Fin de Estudios (TFE). La aplicación para la programación de tareas, detallada en este documento, será evolucionada y utilizada por NTT Data con fines comerciales.

La aplicación se basa en una arquitectura de microservicios. Esta arquitectura permite la integración eficiente de diversas tecnologías adaptadas específicamente para cada uno de los objetivos propuestos. Esta estructura modular optimiza la flexibilidad y la escalabilidad del sistema, por lo que soportará la carga de distintos usuarios. También facilita el mantenimiento independiente de cada componente.

La red de microservicios incluye un componente central para la programación de tareas en segundo plano utilizando servicios de computación en la nube. Además, se integran microservicios adicionales para la gestión de los diversos usuarios y la visualización de los registros generados por las tareas ejecutadas en la nube.

Para mejorar la experiencia del usuario, se ha diseñado una interfaz intuitiva y amigable que simplifique significativamente la interacción con la aplicación, haciendo más accesible y eficiente la programación de las tareas.

Además, se integran técnicas de Inteligencia Artificial (IA) para asistir a los usuarios en la programación de tareas. Esto incluye el desarrollo de un asistente inteligente que permitirá a los usuarios definir las programaciones de manera eficiente y adaptativa en el entorno de la nube.

Como principal resultado del proyecto, se ha obtenido una aplicación web concebida para que, potencialmente, los clientes de NTT Data puedan beneficiarse de la programación de tareas, liberándolos de las complejidades técnicas y permitiéndoles centrarse en la definición de sus tareas.

Agradecimientos

Quiero expresar mi más profundo agradecimiento a mis padres y hermanas por acompañarme a lo largo de esta etapa y brindarme su incondicional apoyo y ayuda en todo momento.

A mis amigos, gracias por estar siempre ahí para mí, a pesar de que no hayamos podido vernos con frecuencia en los últimos meses. Su apoyo ha sido fundamental.

A todas las personas que conocí en Ciudad Real, tanto dentro como fuera de la escuela, les agradezco por haber hecho esta etapa más amena y divertida. Su compañía ha sido invaluable.

Finalmente, un agradecimiento especial a David y Pedro por permitirme realizar este TFG junto a ellos y por sus valiosos consejos y apoyo. Sin su colaboración, esto no hubiera sido posible.

Mohamed

A mi familia

Índice general

Índice de tablas

Índice de figuras

Índice de listados

Listado de acrónimos

IAAS	Infrastructure as a Service
PAAS	Platform as a service
SAAS	Software as a service
IPAAS	integration Platform as a Service
AWS	Amazon Web Services
AWS-EB	Amazon Web Services EventBridge
GCSCH	Google Cloud Scheduler
GCTK	Google Cloud Tasks
ALA	Azure Logic Apps
KVM	Kernel Virtual Machine
DLQ	Dead-Letter Queue
URL	Uniform Resource Locators
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
API	Application Programming Interface
REST	Representational State Transfer
SOA	Service-oriented architecture
AMQP	Advanced Message Queuing Protocol
VPS	Virtual Private Server
DSS	Decision Support System
GDSS	Group Decision Support System
EIS	Enterprise Information System
KD-DSS	Knowledge-Driven Decision Support System
BI	Business Intelligence
IDE	Entorno de Desarrollo Integrado
CRUD	Create, Read, Update and Delete
LLM	Large Language Model
UI	User Interface
K8S	Kubernetes
SC	Storage Class
PV	Persistent Volume
PVC	Persistent Volume Claim

DNS	Domain Name System
JAR	Java ARchive
MVC	Model–view–controller
JPA	Java Persistence API
IoC	Inversion of Control
NLP	Natural Language Processing
TFG	Trabajo Fin de Grado
FORTE	FORTalecimiento de la Empleabilidad
IA	Inteligencia Artificial
ESI	Escuela Superior de Informática
TFE	Trabajo Fin de Estudios

Capítulo 1

Introducción

ESTE capítulo introduce el proyecto desarrollado a lo largo del presente Trabajo Fin de Grado (TFG), y que se discute en detalle en el resto del documento. Primero, se expone el contexto del desarrollo del proyecto. A continuación, se describe el problema a resolver y la solución propuesta. Por último, se describe la estructura del documento.

1.1 Contexto del Proyecto

Este proyecto nace como un convenio FORTE, una de las líneas de acción del programa profESIonalízate lanzado por la ESI de Ciudad Real. Este programa busca fortalecer las competencias profesionales de los egresados de la ESI mediante convocatorias anuales.

El objetivo principal de los convenios FORTE es permitir que los estudiantes próximos a finalizar su carrera o máster, idealmente solo pendientes del TFE, se involucren en el desarrollo de un proyecto real dentro de una empresa y realicen su TFE en ese contexto. Esto ofrece la oportunidad de combinar prácticas empresariales y el desarrollo del TFE, siendo tutorizados tanto por un profesor de la ESI como por un ingeniero de la empresa.

Esta iniciativa permite a los estudiantes integrarse en el mundo empresarial antes de finalizar su carrera, asumiendo desde el inicio mayores responsabilidades en comparación con las prácticas tradicionales.

Este proyecto FORTE ha sido realizado en colaboración con NTT Data, una de las principales empresas globales de servicios de tecnología de la información. NTT Data se especializa en brindar soluciones y servicios tecnológicos innovadores a una amplia gama de industrias, incluyendo la banca, la salud, las telecomunicaciones y el sector público. Con una presencia global y una reputación consolidada, NTT Data se posiciona como un líder en el desarrollo e implementación de tecnologías avanzadas.

La colaboración con NTT Data ha aportado una valiosa oportunidad de trabajar en un entorno profesional de alto nivel, donde ha sido posible aplicar los conocimientos adquiridos en la universidad a proyectos reales y desafiantes. Además, la tutoría proporcionada por ingenieros experimentados de NTT Data ha ofrecido una visión práctica y estratégica del desarrollo del proyecto, lo que ha resultado en un beneficio para su progreso.

1.2 Descripción del problema

En diversos proyectos de NTT Data, los clientes requieren añadir una funcionalidad crítica: la programación de tareas en segundo plano. Este tipo de tareas permite la ejecución de programas de manera diferida según una planificación específica. Las tareas en segundo plano pueden incluir operaciones como el procesamiento de grandes volúmenes de datos, el envío de correos electrónicos o la generación de informes. Estas tareas son esenciales para muchas aplicaciones que necesitan procesar operaciones de manera asíncrona, permitiendo que las funciones principales del sistema continúen sin interrupciones y sin sobrecargar el sistema en tiempo real.

Los clientes desean implementar esta capacidad sin aumentar la carga computacional de sus servidores principales. Mantener la eficiencia y el rendimiento del sistema sin la necesidad de incrementar recursos adicionales es un factor crítico. Aumentar la capacidad de los servidores para soportar estas tareas podría resultar costoso y, en algunos casos, inviable. Por lo tanto, es necesario encontrar una solución que pueda manejar estas operaciones de manera eficiente sin comprometer el rendimiento del sistema principal.

Desde la perspectiva de NTT Data, crear múltiples desarrollos independientes para esta funcionalidad resulta costoso y poco eficiente. La duplicación de esfuerzos y los altos costos asociados con el mantenimiento de soluciones fragmentadas son problemas significativos. Además, mantener consistencia y calidad a través de múltiples desarrollos es un desafío. Es primordial encontrar una solución centralizada y unificada que permita gestionar la programación de tareas en segundo plano de manera eficiente y escalable para todos los clientes.

1.3 Solución propuesta

La solución propuesta, y que se discute en profundidad en el capítulo 5, consiste en el desarrollo de una aplicación centralizada basada en una arquitectura de microservicios diseñada para la programación de tareas en la nube. Esta plataforma única está diseñada para satisfacer las necesidades de diversos proyectos, eliminando la necesidad de crear módulos específicos para cada cliente. Centralizar esta funcionalidad ofrece una serie de beneficios significativos para NTT Data y sus clientes.

Para la aplicación propuesta, se sugiere la implementación de cinco microservicios esenciales que permitan una programación de tareas eficiente y una experiencia de usuario optimizada. A continuación, se detalla la estructura y funcionalidad de cada uno de estos microservicios.

Además de la flexibilidad tecnológica, centralizar la programación de tareas en segundo plano reduce considerablemente los costos y el esfuerzo asociado con el desarrollo y mantenimiento de múltiples soluciones fragmentadas. Al evitar la duplicación de esfuerzos, los recursos de desarrollo pueden ser optimizados y reenfocados en otras áreas.

Otro beneficio clave es la mitigación del impacto en el rendimiento del sistema principal. Al operar de manera independiente, la carga generada por las tareas en segundo plano no afecta la ejecución de otras aplicaciones críticas. Esto asegura que los servicios ofrecidos a los clientes mantengan altos estándares de rendimiento y eficiencia, fortaleciendo la reputación de NTT Data como proveedor de soluciones tecnológicas confiables y robustas.

En primer lugar, el microservicio de programación de tareas es el componente central de la solución. Su principal función es permitir la programación de tareas, y para ello se conecta a un servicio de computación en la nube.

El segundo componente es el microservicio de gestión de usuarios, encargado de todas las operaciones relacionadas con los usuarios, incluyendo la creación, actualización y eliminación de perfiles. Además, maneja la autenticación y autorización, garantizando un acceso seguro y personalizado a la aplicación.

El tercer microservicio es el microservicio de registros de eventos, diseñado para capturar y almacenar los registros de los eventos generados por las tareas. Este microservicio permite el seguimiento detallado de las acciones generadas por las tareas programadas, demostrando así el funcionamiento de las tareas.

El cuarto componente, el microservicio de interfaz de usuario, gestiona la interfaz con la que los usuarios interactúan con la aplicación. Se centra en proporcionar una experiencia de usuario intuitiva y fluida, mejorando la accesibilidad y usabilidad de la aplicación.

El quinto y último componente es el microservicio de apoyo, desarrollado con técnicas de Inteligencia Artificial (IA). La integración de la IA en el desarrollo de software ha revolucionado la manera en que se diseñan y ejecutan las aplicaciones. Este microservicio incluye un sistema de IA que actúa como asistente inteligente para la programación de tareas, ayudando a los usuarios a definir planificaciones que se ajusten a sus requisitos de manera más efectiva.

Se han investigado diversas técnicas para proporcionar apoyo al usuario en este microservicio, tales como sistemas inteligentes y sistemas de aprendizaje automático. Como se discutirá más adelante en este documento, se optó por implementar un sistema de IA, específicamente un subcampo del aprendizaje automático conocido como LLM, el cual, se adapta mejor al usuario. Este componente es capaz de entender lenguaje natural utilizado por los usuarios, facilitando así una interacción sencilla y valiosa para ellos.

El desarrollo de este componente ha supuesto el estudio profundo de las técnicas de aprendizaje automático y la optimización en el entrenamiento del modelo. Ha sido necesario entrenarlo con un conjunto de datos para adecuar este modelo general al caso específico de esta aplicación.

La funcionalidad del microservicio de apoyo se centra en proporcionar recomendaciones y sugerencias inteligentes basadas en el contexto de las tareas definidas por los usuarios. Utilizando el sistema de IA implementado, los usuarios pueden interactuar de manera natural y obtener planes de acción optimizados para sus actividades diarias.

Se pretende desplegar la aplicación para que sea accesible desde Internet. Esto implica utilizar un orquestador de microservicios para la gestión, proporcionando capas de seguridad y robustez. Posteriormente, se expondrá la aplicación a Internet. Este último paso pondrá la aplicación en funcionamiento, demostrando su operatividad como un paso intermedio hacia su despliegue en un entorno profesionalizado de producción.

Para finalizar, se introducen los beneficios de utilizar la arquitectura de microservicios.

- Es una arquitectura flexible, que permite la integración de diversas tecnologías de manera eficiente. Esto significa que la plataforma puede adaptarse y evolucionar con tecnologías emergentes sin comprometer la estabilidad del sistema general.
- Los microservicios son independientes, lo que proporciona un sistema más tolerante a fallos. Si uno de ellos falla, el sistema podrá seguir siendo utilizado. Además, esta independencia facilita el mantenimiento de la aplicación, ya que cada microservicio puede ser actualizado o reparado de manera aislada sin afectar al resto del sistema.

1.4 Estructura del documento

Capítulo 2: Objetivos

En este capítulo se presentan los objetivos principales y específicos del proyecto.

Capítulo 3: Estado del arte

En este capítulo se exploran los fundamentos teóricos y las tecnologías estudiadas para el diseño e implementación del proyecto.

Capítulo 4: Metodología

Este capítulo detalla la metodología seguida durante el proyecto, así como los recursos utilizados.

Capítulo 5: Arquitectura

Este capítulo describe la arquitectura del sistema implementado. También se explican los problemas que surgieron durante el proyecto y las soluciones consideradas.

Capítulo 6: Resultados

Este capítulo presenta los resultados finales del proyecto.

Capítulo 7: Conclusiones

Este capítulo evalúa los objetivos logrados y propone mejoras para el sistema.

Capítulo 2

Objetivos

ESTE capítulo describe los objetivos a abordar durante el desarrollo del presente TFG. Primero se presentan el objetivo general y, a continuación, los objetivos parciales que han servido para alcanzar el objetivo general.

2.1 Objetivos generales

El objetivo principal de este TFG es el diseño y desarrollo de una aplicación con la que se puedan programar tareas en una plataforma de computación en la nube. Esta aplicación deberá abstraer al usuario de la plataforma elegida así pues se necesita darle la mayor de las facilidades al usuario.

La aplicación debe ser diseñada de manera sencilla y agradable, asegurando una experiencia de usuario óptima. Para lograr esto, la interfaz debe ser intuitiva y amigable, permitiendo a los usuarios navegar y realizar tareas sin complicaciones. Además, se le debe proporcionar ayuda en la definición de las tareas en segundo plano, evitando que necesite conocimiento previo para su definición. Además, la aplicación deberá gestionar distintos niveles de permisos para los usuarios, cada uno con diferentes grados de acceso a las funcionalidades de la aplicación.

Por último, se pretende desplegar la aplicación en Internet para realizar demostraciones de su funcionamiento. Este despliegue permitirá que cualquier usuario pueda acceder y probar la aplicación, evaluando sus capacidades y proporcionando retroalimentación valiosa para futuras mejoras.

2.2 Objetivos parciales

Los objetivos parciales definidos para alcanzar el objetivo general son los siguientes:

- **Estudio comparativo de las principales plataformas de computación en la nube y del soporte que ofrecen para tareas en segundo plano.** Es necesario emplear una primera etapa en el estudio de las plataformas de computación en la nube, obteniendo una base de conocimiento para el desarrollo de la aplicación. También se debe elegir la plataforma ideal para este desarrollo, teniendo en cuenta varios parámetros como sus prestaciones, tolerancia a fallos o coste.

- **Diseño y desarrollo del microservicio que permita la programación de tareas.** Se debe diseñar y desarrollar el microservicio que permita la definición de tareas en una plataforma de computación en la nube. El microservicio debe abstraer al usuario de la plataforma de computación en la nube elegida y debe permitirle tanto la creación como la modificación de las tareas.
- **Diseño y desarrollo de los microservicios auxiliares.** Se debe diseñar y desarrollar los microservicios que sirvan como apoyo a la programación de tareas en la nube. Estos microservicios auxiliares se encargarán de la gestión de los usuarios, de mostrar los eventos generados por la tarea y por último la interfaz con la que el usuario interaccionará.
- **Desarrollo de un módulo de apoyo a la toma de decisiones relativo al perfilado de horarios.** Una vez se han desarrollado los microservicios esenciales para la aplicación, también es interesante añadir un módulo que facilite la programación de tareas. Se debe desarrollar un microservicio que facilite la programación de tareas en la nube de tal forma que al usuario le resulte agradable utilizar este módulo.
- **Despliegue de los microservicios.** Se debe desarrollar una estrategia que facilite la transición entre la fase de desarrollo y la de despliegue de los distintos microservicios de la aplicación. Por otro lado, se debe diseñar el método de despliegue de los microservicios garantizando el acceso a la aplicación mediante Internet.
- **Testing y validación de la aplicación.** Se probarán los distintos módulos de la aplicación con el fin de validar su funcionamiento y asegurarse de que el sistema es tolerante a fallos. También se probará el asistente virtual para asegurarse de que las expresiones cron generadas son las esperadas. Por último se validará la interfaz y se modificará en caso de que sea necesario para adaptarla a los usuarios.

Capítulo 3

Estado del arte

EL del estado del arte tiene como objetivo ilustrar los avances tecnológicos que han sido fundamentales para el desarrollo de este proyecto.

En primer lugar, se realiza un estudio de la gestión de tareas en segundo plano, trazando su evolución desde sus orígenes hasta su actual implementación en la nube. Este análisis nos permitirá entender mejor cómo se han adaptado y optimizado estos procesos para su uso en entornos de nube.

Posteriormente, se explorará el paradigma de la computación en la nube, examinando conceptos clave y analizando la interrelación de sus componentes para dar forma a esta tecnología.

Finalmente, se presentará una serie de tecnologías relevantes que han sido utilizadas durante el desarrollo del proyecto. Se proporcionará el contexto necesario para comprender su funcionamiento y su contribución al logro de los objetivos del proyecto.

3.1 Gestión de Tareas en segundo plano

3.1.1 Definición

La gestión de tareas en segundo plano es la ejecución de procesos y operaciones sin interacción directa del usuario [IKKO14]. Este tipo de tareas son cruciales para aumentar la eficiencia de las aplicaciones, ya que permite que el hilo principal este libre para otros procesos que sean más importantes como pueden ser la carga de interfaces o de datos para un usuario. Este enfoque evita que el usuario deba esperar reduciendo su interés por la aplicación que puede llegar a tener un rendimiento pésimo o incluso se bloquea. De esta manera procesos no tan relevantes como reproducir música o enviar un correo a una hora determinada se pueden realizar en segundo plano aumentando la eficiencia.

Existen diversos tipos de tareas en segundo plano dependiendo de la naturaleza y el momento de ejecución de esta.

- *Tarea transitoria*: Una aplicación que realiza una tarea corta y que debe realizarse en segundo plano hasta que se complete manteniendo el hilo principal libre para otros procesos. Casos comunes son la compresión de datos o el procesamiento de archivos.

- *Tarea Continua*: Son tareas de mayor duración que se ejecutan en segundo plano y que pueden ser iniciadas expresamente por el usuario. Algunos ejemplos son la reproducción de música o el navegador GPS.
- *Tarea Programada*: Tareas que se ejecutan de manera asíncrona en un momento posterior en el tiempo definido por el usuario. Este tipo de tareas se ejecutan sin que el usuario sea consciente de ello. Algunos tipos son la sincronización de bases de datos o el análisis del dispositivo en busca de virus.

Este TFG se centrará en el último tipo, en las tareas programadas o scheduled works. Este tipo de tareas son utilizadas por los desarrolladores para programar la realización de eventos en un momento posterior de manera asíncrona, sin que el usuario que esté utilizando el dispositivo sea consciente de dicha tarea. Son de mucha utilidad para las empresas, permitiéndoles automatizar procesos, ahorrando costes y definiéndolos en tiempos donde la carga de la aplicación sea más baja, mejorando el rendimiento de la aplicación.

Casos de Uso

Este tipo de tareas tienen muchas aplicaciones prácticas dentro de las empresas. Algunos de los casos de uso principales se comentarán a continuación[1].

Bases de Datos: Las bases de datos copan la mayoría de las funcionalidades de las tareas programadas. Existe una gran cantidad de servicios que se pueden integrar para mejorar el funcionamiento de la base de datos. Algunos de los casos de empleo incluyen:

- Creación de copias de seguridad
- Mantenimiento de la base de datos, con limpiezas y migraciones de datos de manera regular para mantener la consistencia e integridad de los datos.
- La sincronización de bases de datos también es otro proceso que se automatiza con este enfoque para asegurar la disponibilidad y respuestas ante errores.

Otros procesos: Otros procesos para las tareas programadas incluyen la generación de reportes para los stakeholders del proyecto o la escala de recursos para una aplicación, de modo que se aumentan los recursos en las horas punta de trabajo y se liberan estos recursos cuando la carga sea de menor intensidad.

También es común el envío de mensajes por correo electrónico planificando previamente la fecha de envío. La mayoría de los servicios de mensajería actuales integran este servicio en sus plataformas y es ampliamente utilizado.

[1] `https://runcloud.io/blog/cron-jobs`

3.1.2 Administradores de Tareas

En este apartado se estudiará una serie de administradores de tareas, centrandose en los primeros que se implementaron y cuáles son sus características.

Cron

Cron fue el inicio de la programación de tareas en una computadora, cron es un comando de sistemas operativos UNIX y Linux que te permite programar tareas [LL21]. Cron está pensado para ejecutar comandos en intervalos de tiempo definidos por el usuario. Para ello se especifica el tiempo con una sintaxis específica (figura 3.1 y tabla 3.1) y esta información se almacena en un archivo llamado “crontab”. Cron funciona como un daemon o servicio que ejecuta las tareas en segundo plano, para ello, cada minuto consulta las crontabs y determina si tiene que realizar algún proceso.

La sintaxis de cron es la siguiente:

```
|------------------------------- Minute (0-59)
|     |------------------------- Hour (0-23)
|     |     |------------------- Day of the month (1-31)
|     |     |     |------------- Month (1-12; or JAN to DEC)
|     |     |     |     |------- Day of the week (0-6; or SUN to SAT; or 7 for Sunday)
|     |     |     |     |
|     |     |     |     |
*     *     *     *     *
```

Figura 3.1: Sintaxis cron. Imagen obtenida de: Google formato cron

Cuadro 3.1: Configuración de tareas cron

Símbolo	**Significado**	**Ejemplo**
*	Selecciona todos los posibles valores	En el campo minuto indica que se ejecutará cada minuto
,	Separar una serie de valores	"1,2.^{en} el campo día de la semana, es lunes y martes
-	Intervalo de valores	"1-5.^{en} el campo día de la semana, es de lunes a viernes
*/	Definir un salto en la frecuencia	"*/5.^{en} el campo minuto indica que se ejecutará cada 5 minutos
L	Se utiliza en el día y mes para indicar el "último"valor	"1L.^{en} el campo día del mes, es el último lunes del mes
W	Se utiliza en el día para indicar el valor más cercano	"15 W.^{en} el campo día del mes, es el día de la semana más cercano al día 15 del mes
#	Se utiliza para indicar el cardinal de la ocurrencia de una semana del mes	"6#3.^{en} el campo día de la semana, es el tercer viernes del mes

Para definir la recurrencia de la tarea puedes precisar hasta el minuto, e incluso puedes definir rangos de valores, e intervalos de tiempo entre dos fechas. Por ejemplo, si se quiere ejecutar una tarea todos los lunes y viernes a las 16:30, su sintaxis sería: (30 16 * * 1,5). Por otro lado, para ejecutar una tarea cada 5 minutos entre las 2 y las 3 del primer día del mes (*/5 2 1 * *). Aunque su sintaxis es compleja, ofrece una gran variedad de valores e incluso existen herramientas online que facilitan la obtención de la frecuencia deseada.

Anacron

La principal desventaja de cron es que necesitas que el dispositivo esté iniciado para que se puedan ejecutar estas tareas y si el dispositivo no se encuentra disponible, esa tarea se perderá. Para resolver dicha limitación surge anacron como complemento a cron y en este caso, la herramienta se centra en dispositivos que no están encendidos todo el tiempo [LL21]. Anacron sigue los mismos principios que cron, por ejemplo, implementa el mismo formato de tablas (anacrontabs) sin embargo, si el dispositivo no está encendido en el momento en el que se debería ejecutar una tarea, en cuanto el dispositivo arranque, esta tarea se ejecutará.

Anacron está diseñado para tareas que sean diarias, pero que no importa la hora del día en el que se realicen, por ello, una de sus limitaciones es que solo puede especificarse las tareas hasta el nivel de día como máximo. Anacron a diferencia de cron, no es un daemon, en este caso se ejecuta durante el inicio del sistema o por otros programadores como puede ser cron.

At

En el caso de que se quiera planificar una tarea a futuro que se ejecutará en otro momento una única vez, cron no sería una opción recomendable, ya que deberá ser el usuario quien manualmente elimine la tarea. Es por ello por lo que existe como alternativa “at”. Este comando de sistemas Linux, ejecuta comandos especificando la fecha y hora mediante un timestamp [LL21].

Otras implementaciones

A raíz de cron, surgieron nuevas implementaciones para tratar de mejorar cron. Una de estas implementaciones es *vixie-cron* [YDM02] desarrollado por Paul Vixie y que implementa algunas funcionalidades nuevas, como por ejemplo la definición del usuario que ejecutará la tarea. Distribuciones como Debian tienen su planificador de tareas basado en vixie-cron. Otra de las implementaciones es *fcron* [CA07], un programador cuyo objetivo es sustituir a vixie-cron, implementa la mayoría de sus funcionalidades y además también la característica principal de anacron, ejecutar tareas en sistemas que no están siempre operativos, también permite ejecutar flujos de tareas y personalizaciones como ejecutar tareas estableciendo condiciones como la carga de la CPU.

3.2 Computación en la nube

3.2.1 Definición

La computación en la nube es un paradigma computacional en el que un proveedor de servicios ofrece recursos computacionales a sus clientes bajo demanda [WB10]. La computación en la nube se ha convertido en una solución que potencia el crecimiento de las empresas [MLB+11], implementando las economías de escala en este paradigma, puesto que permite que empresas sin importar su tamaño o sector puedan acceder a una infraestructura de computación bajo demanda y de gran interés para muchas aplicaciones.

Empresas que tengan ideas innovadoras no necesitan adquirir una gran cantidad de recursos para poder probar sus soluciones, pueden probar dichas soluciones utilizando la infraestructura y servicios de la nube y basándose en sus resultados pueden plantear la adquisición de los dispositivos para crear una nube privada.

La nube se ha utilizado como un término general para describir la red de Internet. No se debe confundir con una entidad física, sino que es una red global de servidores alrededor del planeta que operan interconectados como un ecosistema único. La computación en la nube, por tanto, funciona como una extensión de los servicios en la web, es decir, se ofrecen servicios de computación a los que se puede acceder a través de Internet, por ejemplo, a través de un navegador.

La computación en la nube es una estrategia ganadora para cualquier empresa otorgando una serie de ventajas con respecto a otros paradigmas [QLDG09]:

- Elimina la necesidad de comprar y mantener dispositivos físicos por parte de la empresa, lo cual puede llevar a un ahorro en los costes iniciales de los proyectos.
- Es una tecnología accesible desde cualquier lugar con conexión a internet
- Mejora la eficiencia de los dispositivos gracias a su planificador de recursos
- Ofrece servicios a clientes bajo demanda, ofreciéndoles elasticidad en los recursos físicos y de software para que estos aumenten o disminuyan cumpliendo con los requisitos computacionales de sus aplicaciones.

Funcionamiento teórico

La base para poder desarrollar los servicios en la nube es la virtualización [XZ12]. La virtualización es un proceso que permite a un dispositivo compartir sus recursos de hardware como piezas únicas separadas de forma digital, este proceso divide los recursos de hardware de un único dispositivo en unidades más pequeñas, cada una con una división de la memoria y capacidad de procesamiento del dispositivo original [PC20]. Estas unidades se denominan máquinas virtuales.

Además, se necesita un componente software que administre una serie de máquinas virtuales en la misma máquina física, este componente es el hipervisor (figura 3.2). El hipervisor [AEB16] asigna los recursos de computación a las distintas máquinas virtuales según su demanda. El hipervisor puede ser de dos tipos dependiendo del lugar de instalación, si se instala en el hardware de la computadora es de tipo 1, también conocido como hipervisor bare metal y tiene acceso a los recursos de hardware. Por otro lado, si se instala en el sistema operativo es de tipo 2, también conocido como hipervisor alojado.

Las máquinas físicas equipadas con un hipervisor de tipo 1, como la KVM es la máquina host. Suele utilizarse este hipervisor de tipo 1, son más eficientes y escalables, además de que son más seguros debido a que no tienen un sistema operativo que pueda ser atacado.

Las máquinas virtuales están equipadas con un hipervisor de tipo 2, estas utilizan los recursos que le asigna la máquina host e incluso es capaz de hacer peticiones al host para ampliar sus recursos, se llaman máquinas guest. Son más fáciles de utilizar, pero menos eficientes.

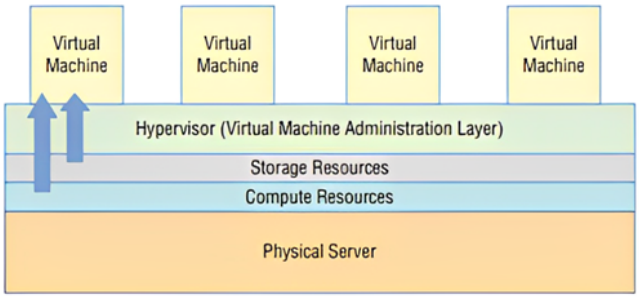

Figura 3.2: Máquina virtual host. Imagen obtenida de: [PC20].

En la nube, la arquitectura que se sigue es la misma, se tiene una serie de recursos de hardware en un centro de datos que son administrados por un hipervisor de tipo 1, el cual administra máquinas virtuales que pueden ser compartidas por varias personas a la vez gracias a la tenencia múltiple o multitenancy.

La tenencia múltiple o multitenancy [AAG+14] es un modelo de software en el que se comparten los mismos recursos de un servidor para una serie de clientes, sin que estos sean conscientes de ello y manteniendo la privacidad y seguridad de sus datos mediante el aislamiento. Este modelo permite ofrecer servicios a escala y a un costo reducido, ya que los recursos se pueden asignar dependiendo de las necesidades de los inquilinos. Los servicios de computación en la nube ofrecen una serie de recursos físicos y virtuales conocidos como agrupación de recursos o resource pooling que son asignados a las máquinas virtuales de manera dinámica.

Servicios principales de la computación en la nube

La computación en la nube ofrece principalmente tres tipos de modelos de servicio [ST20]: Infrastructure as a Service (IAAS), Platform as a service (PAAS) y Software as a service (SAAS). Estos modelos representan distintos enfoques en la entrega de recursos de computación (figura 3.3).

- IAAS: Es el nivel que te otorga el mayor control del servicio que adquieres y se centra en el alquiler de recursos de la infraestructura cloud como pueden ser máquinas virtuales, servidores o dispositivos de almacenamiento.
- PAAS: Es un entorno que te ofrece el hardware y software necesario para el desarrollo y despliegue de aplicaciones en la nube pagando solo los recursos que necesitas para que te centres en el desarrollo.
- SAAS: SaaS te ofrece una aplicación de usuario final desarrollada y mantenida por el proveedor de la nube con diversos objetivos como la comunicación (email) o aplicaciones que aumentan la productividad (Office 365).

Figura 3.3: Tipos de servicio cloud. Imagen obtenida de: Microsoft Azure

Despliegues en la computación en la nube

Además, la computación en la nube ofrece una variedad de despliegues para satisfacer las necesidades específicas de cada organización, existen tres tipos de despliegues [AKM18]:

- *Nube pública*: alojadas y gestionadas por proveedores de servicios en la nube que entrega los servicios informáticos a través de internet bajo demanda. Te ofrece alta escalabilidad y reduce los costes iniciales de montar la infraestructura necesaria para utilizar servicios en la nube.
- *Nube privada*: alojar y gestionar la infraestructura informática localmente dentro de las instalaciones de una organización (se conocen comúnmente como on-premises). Incluye desde la gestión de datos hasta la implementación de la seguridad. Otorga un mayor control y personalización para satisfacer las necesidades de la empresa.

- *Nube híbrida*: es un entorno de computación mixta donde las aplicaciones utilizan una combinación de recursos alojados en las nubes públicas y privadas. Son las más extendidas porque en la práctica no se depende de una única nube para alojar todos los servicios necesarios, si no que se combinan varias nubes públicas y privadas para asegurar la calidad del servicio e implementar medidas de seguridad propias de la compañía.

3.2.2 Comparación de Plataformas en la nube

Los sistemas en la nube ofrecen como uno de sus servicios la gestión de trabajos en la nube, abarcando los distintos tipos de trabajos que existen y las distintas posibilidades en su gestión.

Amazon Web Services

AWS-EB[2] es un servicio de Amazon Web Services (AWS) diseñado para la gestión de tareas en la nube [PC20]. Su principal función es programar tareas para que se ejecuten de manera asíncrona, permitiendo así que las tareas se realicen en un momento distinto al de su invocación. Este enfoque es especialmente útil para evitar la sobrecarga de un sistema y para llevar a cabo procesos en segundo plano durante periodos de menor carga de trabajo. Existen principalmente dos métodos para utilizar AWS-EB: uno basado en reglas y otro en programación. Ambos métodos se explicarán a continuación.

Un aspecto importante a tener en cuenta es que AWS-EB garantiza la entrega de cada evento al menos una vez. Esto implica que si un evento desencadena una operación, puedes estar seguro de que dicha operación se ejecutará al menos una vez. Sin embargo, en situaciones de alta carga o fallos de red, un evento puede entregarse más de una vez. Por lo tanto, es crucial que las operaciones desencadenadas por eventos sean idempotentes, es decir, que el efecto de realizar la misma operación varias veces sea el mismo que realizarla una sola vez. Esto garantiza que tu sistema funcione correctamente incluso si un evento se entrega varias veces.

Además, AWS-EB ofrece la posibilidad de definir mecanismos para la recuperación de errores. Estos mecanismos realizan reintentos con un "backoff" exponencial[3], lo que significa que el intervalo de tiempo entre reintentos aumenta con cada intento fallido (figura 3.4). También puedes definir una Dead-Letter Queue (DLQ) para no perder los eventos que no han conseguido llegar a su destino con éxito.

Finalmente, AWS-EB se centra en llamar a agentes externos que realizarán sus funciones. Por lo tanto, AWS-EB garantiza la llamada al agente, pero si el agente falla en su proceso, este fallo no será registrado por el servicio.

[2] `https://aws.amazon.com/es/eventbridge/`

[3] `https://docs.aws.amazon.com/prescriptive-guidance/latest/cloud-design-patterns/retry-backoff.html`

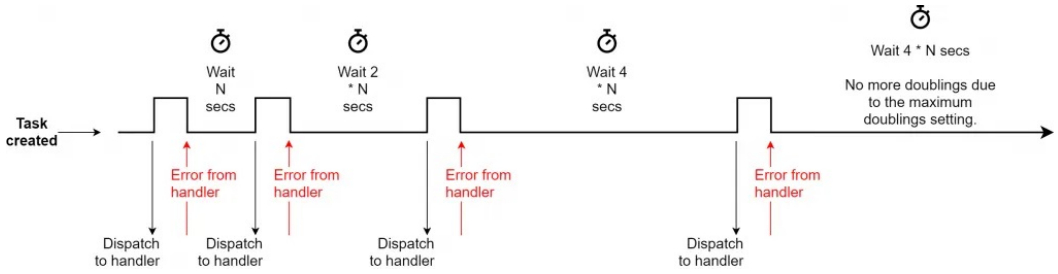

Figura 3.4: Backoff exponencial. Imagen obtenida de: Medium Scheduling Tasks

Basado en reglas

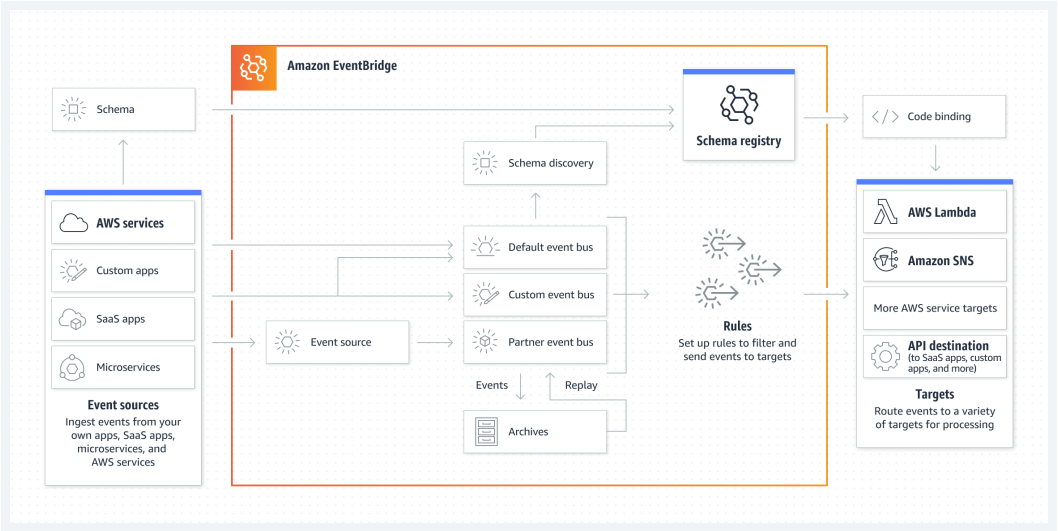

Figura 3.5: AWS-EB basado en reglas. Imagen obtenida de: AWS EventBridge

Existen varios servicios que pueden enviar tareas a AWS-EB. Estos pueden ser aplicaciones propias que se conectan al cliente de Amazon, aplicaciones de AWS o partners SAAS como Salesforce o Auth0. Estos servicios envían tareas a un bus de destino.

Los buses de destino actúan como un router, almacenando tareas para entregarlas a su respectivo destino en el momento adecuado. Este tipo de buses se utilizan cuando se desea recibir tareas desde uno o varios orígenes y entregarlas a uno o varios destinos (desde 1 hasta 5 destinos).

En estos buses se configuran reglas (figura 3.5) para determinar los destinos de las tareas. Existen dos tipos de reglas: las basadas en patrones y las basadas en planificación. Las reglas basadas en patrones verifican que el evento recibido cumpla ciertas condiciones, como la aplicación de origen o campos personalizados enviados en el payload del evento. Por otro lado, las tareas basadas en planificación permiten distintas configuraciones. Pueden ejecutarse utilizando el planificador de cron o un planificador basado en repetición para definir tareas que se ejecuten periódicamente.

El planificador de cron ofrece una planificación más detallada y flexible de la repetición de los eventos, permitiendo indicar desde la ejecución el primer lunes del mes hasta la repetición cada hora. El planificador basado en repetición es más sencillo y solo permite indicar la frecuencia de repetición, por ejemplo, cada 5 minutos o cada día. Sin embargo, con ambos tipos de planificación no se puede definir la fecha de inicio ni de fin de los trabajos. Una vez creada la regla, esta comenzará a enviar trabajos y será responsabilidad del usuario crear y eliminar la regla en el momento adecuado.

Finalmente, existen muchos destinos posibles para las tareas, con el fin de satisfacer todas las necesidades del cliente. Estos pueden incluir una llamada a una Application Programming Interface (API) o la utilización de diferentes servicios de AWS, como una función de lambda, un topic de Amazon Simple Notification Service de Amazon para enviar notificaciones o una tarea de Amazon Elastic Container Service.

Programados

Figura 3.6: AWS-EB Scheduler. Imagen obtenida de: AWS EventBridge

Además de las tareas basadas en reglas, AWS-EB permite programar eventos (figura 3.6). Este tipo de tareas ofrece una mayor flexibilidad en la programación. Se basa en la funcionalidad de las reglas programadas, pero ofrece más funcionalidades, por ejemplo, una de las principales diferencias con las tareas basadas en reglas es que puedes definir una tarea para que se ejecute una única vez. También puedes definir tareas utilizando expresiones cron o por repetición.

Otra de las ventajas de este enfoque es la posibilidad de definir la zona horaria de la tarea o una ventana de tiempo para la ejecución. Por ejemplo, si defines una ventana de 15 minutos, esto indica que una tarea se realizará dentro de un margen de 15 minutos desde el momento indicado para la ejecución de la tarea. Esta característica es especialmente útil cuando tienes muchos microservicios y quieres evitar la congestión de estos en periodos de alta demanda.

Además, puedes definir una fecha de inicio y de fin para la repetición de las tareas, lo que te permite tener un control más preciso sobre cuándo se ejecutan tus tareas. Sin embargo, una limitación de este enfoque es que solo se puede llamar a un único destino. Además, no se puede llamar a una API externa directamente, sin embargo, puedes llamar a la API utilizando como intermediario uno de los servicios de AWS como es AWS Lambda.

Google Cloud

Google Cloud también ofrece un servicio para la gestión de tareas en la nube [MV23]. En su caso, Google Cloud ofrece dos servicios dependiendo de las especificaciones de la tarea que se quiere llevar a cabo. Google Cloud Scheduler (GCSCH)[4] se centra en la programación de tareas recurrentes. Por otra parte, Google Cloud Tasks (GCTK)[5] trata de la realización de tareas ejecutadas una única vez. Comparten con AWS-EB que son eventos que se entregan al menos una vez, por lo que los programas que realices deben ser idempotentes.

Google Cloud Scheduler

GCSCH es la solución de Google Cloud para la programación de tareas recurrentes. Al igual que su contraparte de AWS, AWS-EB Scheduler, GCSCH permite la ejecución de unidades de trabajo en horarios específicos o a intervalos regulares. Este servicio proporciona flexibilidad al permitirte definir tareas utilizando la sintaxis de cron, lo que facilita la configuración de horarios detallados y periódicos para la ejecución de eventos. Además, permite el establecimiento de una fecha de inicio y una fecha de fin. GCSCH garantiza la entrega de eventos al menos una vez, asegurando la ejecución de las tareas programadas.

Una de las principales ventajas de GCSCH es que se te cobra por el trabajo, no por el número de veces que se lanza, por tanto, si se tiene un trabajo, pero se lanzase múltiples veces se te cobrará como un único trabajo. En cuanto a la conectividad, el destino puede ser una llamada a un endpoint externo o la utilización de los servicios propios de la plataforma Google Cloud, es decir, acceder a AppEngine. Además, se te ofrece una posibilidad extra que es utilizar topics de publicador/suscriptor como origen o destino.

Google Cloud Tasks

GCTK es una solución de Google Cloud diseñada para la ejecución asíncrona de tareas, asegurando que cada tarea se ejecute solo una vez. GCTK comparte algunos conceptos con AWS-EB basado en reglas como es la necesidad de una cola donde se almacenarán las tareas. Esta cola tiene una capacidad prácticamente ilimitada, ya que una vez que una tarea alcanza su límite, se ejecuta y se elimina de la cola.

[4] `https://cloud.google.com/scheduler?hl=es-419`
[5] `https://cloud.google.com/tasks?hl=es-419`

GCTk permite definir políticas de reintento con un ”backof” exponencial para manejar los errores en la resolución de eventos. A diferencia de otras soluciones, GCTk no está diseñado para la programación de tareas recurrentes. De hecho, no permite la programación de tareas recurrentes en el tiempo, solo se pueden programar de manera asíncrona hasta un máximo de 30 días. Por lo tanto, GCTk es ideal para acciones como el envío de notificaciones futuras o la realización de actualizaciones puntuales, es decir, procesos que son programados por otros procesos y que se prefieren realizar en otro momento cuando la carga de trabajo sea menor.

Los posibles destinos de GCTk son el acceso a un endpoint HTTP o la llamada a uno de los servicios de AppEngine, no se brinda la opción de utilizar topics de publicador/suscriptor.

Microsoft Azure

Azure, la plataforma de computación en la nube de Microsoft, también integra todos sus servicios de programación, en su caso en el servicio Azure Logic Apps (ALA)[6]. ALA es una integration Platform as a Service (iPaaS) que incluye un servicio de programación de flujos de tareas [MLS20], al contrario que los servicios anteriormente nombrados, en este caso se integra la posibilidad de crear programaciones para flujos de tareas (o bien para una única tarea si se desea). ALA también incluye un servicio para ejecutar tareas una única vez en el futuro, sin limitaciones en el tiempo máximo de espera para esta tarea.

En cuanto al flujo de tareas, se define la recurrencia de la tarea con un formato más simple para el usuario, se asemeja al planificador definido con formato temporal de recurrencia de AWS-EB. Sin embargo, en este caso es mucho más personalizable, además de definir la recurrencia indicando el intervalo (cantidad) y la frecuencia (segundos, minutos, horas, días, semanas o meses), también se puede personalizar para indicar en los días y semanas, a que horas y minutos se quiere realizar la tarea y en las semanas, que días de la semana se quiere realizar esta tarea. Además, se puede definir una fecha de inicio indicando cuando quieres que se realicen las tareas. Esta implementación termina siendo una simplificación de las expresiones cron para que a los usuarios que utilizan la interfaz les sea más agradable definir esta recurrencia, pero, por el contrario, es mucho menos potente. Por ejemplo, con ALA no se puede definir la hora ni el día del mes que quieres que se realice una tarea.

Además, existen dos modos para programar las tareas. Por una parte, existe la recurrencia y, por otro lado, la ventana deslizante. Ambas implementan las mismas funcionalidades, sin embargo, en la recurrencia, si un evento se pierde, este no se recuperará, el programador se iniciará en el siguiente intervalo para continuar con el resto de los eventos. En la ventana deslizante si se recuperan los eventos, por ejemplo, si se pierden cinco eventos por una interrupción del servicio, en cuanto se habilite de nuevo, se lanzarán los cinco eventos atrasados.

[6] `https://azure.microsoft.com/es-es/products/logic-apps`

Como destinos se puede definir llamadas Hypertext Transfer Protocol (HTTP), los propios servicios de Azure e incluso SAAS que te ofrece la plataforma como Twitter (o como se ha renombrado recientemente, X), Gmail o las aplicaciones de Microsoft Office. Al ser un servicio que se implementa orientado al flujo de tareas, se pueden definir una gran cantidad de destinos por cada tarea, en este caso 500 destinos, sin embargo, estos destinos se llamarán de manera secuencial y es posible que si da error en uno de ellos, no se realice la tarea para los siguientes.

Tabla comparativa

Cuadro 3.2: Tabla comparativa de los servicios

	Google Cloud	**AWS EventBridge**	**Azure Logic Apps**
Enfoque	▪ Scheduler: Tareas recurrentes ▪ Tasks: Tareas únicas	▪ Scheduler: Tareas únicas y recurrentes ▪ Reglas: Tareas recurrentes	Tareas únicas y recurrentes
Cron	✓Scheduler ✗Tasks	✓	✗
Rate	✗	✓	✓(Con mayor personalización)
Tiempo de programación	Scheduler: Sin límite Tasks: 31 días	Sin límite	Sin límite
Fecha de inicio	✗	✓Scheduler ✗Reglas	✓
Destino			
HTTP	✓	✓	✓
SaaS	✗	✓	✓
Publicador/ Suscriptor	✓Scheduler ✗Tasks	✓	✓
Máximo número destinos/tarea	1	1 Scheduler 5 por regla	500
Reintentos	▪ Scheduler: Hasta 5 ▪ Tasks: Sin límite	▪ Hasta 185 reintentos ▪ DLQ	✗

Comparación de precios

Uno de los puntos más importantes para decantarse por una de las plataformas para programar tareas es su precio. En este apartado se examinará el precio de cada plataforma y se realizará una aplicación práctica para comparar los precios en cada plataforma, planteando dos casos de uso para dos tipos de empresa de distinto tamaño y mostrando el precio mensual para cada una de las plataformas. Solo se tratará de explicar el coste del programador de tareas que llama a un endpoint HTTP externo que es el que realiza la tarea.

El precio del trabajador que realiza la tarea no se incluye. Por ello, es necesario explicar que en AWS-EB basado en reglas por cada millón de llamadas a una API externa se cobra 0,24 USD por millón, en el resto de las plataformas no se cobra este servicio. Actualmente, no se pueden realizar llamadas con AWS-EB programador por lo que se debe utilizar una función de lambda, en cada caso se crea una función de lambda y se estima su precio con la calculadora de AWS[7].

Precio por plataforma

- AWS-EB[8]:
  - Reglas: 1,00 USD por cada millón de eventos.
  - Programador: 14 millones de invocaciones gratuitas, después, 1,15 USD por cada millón de invocaciones
- Google Cloud:
  - GCSch[9]: 3 tareas gratis al mes (sin importar el número de veces que se invoque), después 0,10 por cada tarea al mes (sin importar el número de veces que se invoque).
  - GCTk[10]: Primer millón gratis, después 5000 millones 0,40 USD por millón.
- Microsoft Azure[11]: En este caso se te cobra por el uso del conector para llamar al proceso que se desencadena y realiza la tarea programada. El coste por llamada es de 0,000125 USD.

[7] https://calculator.aws/#/createCalculator/Lambda
[8] https://aws.amazon.com/es/eventbridge/pricing/
[9] https://cloud.google.com/scheduler/pricing?hl=es
[10] https://cloud.google.com/tasks/pricing?hl=es
[11] https://azure.microsoft.com/es-es/pricing/details/logic-apps/

Casos de Uso

Los siguientes casos de usos tratarán de explicar el precio por tener tareas programadas en los distintos servicios en la nube.

Caso de Uso 1: En este caso se tiene una empresa de gran tamaño que tiene varios servicios que se deben ejecutar de manera asíncrona. Los procesos son los siguientes.

Se puede comprobar que la opción más barata sería utilizar AWS-EB, destacando la programación por reglas para este caso de uso.

Cuadro 3.3: Servicios Caso 1

ID	Servicio	Frecuencia	Número de llamadas al mes
1	Sincronizar base de datos 1	3 veces al día	90
2	Realizar proceso de limpieza	1 vez a la semana	4
3	Copia de seguridad	12 veces al día	360
4	Enviar un correo al cliente	1 vez a la semana	4
5	Recolectar logs	1 vez al día	30
6	Descargar archivos de internet	1 vez al día	30
7	Sincronizar base de datos 2	2 veces al día	60
8	Proceso ETL	1 vez a la semana	4
9	Análisis de seguridad	24 veces al día	720
10	Actualización de una página web	2 veces por semana	8
11	Escalar los servicios de una aplicación	4 veces al día	120
12	Comprobar la disponibilidad de los servicios	24 veces al día	720
		Total:	**2400**

Cuadro 3.4: Precio por servicio Caso 1

Servicio	Precio
AWS-EB reglas	2150 * 1/1.000.000 USD por tarea + llamada al enpoint HTTP (2150 * (0,24/1.000.000 USD)) = 0,00215 + 0,000516 = **0,00266 USD/mes**
AWS-EB programador	14 millones de invocaciones gratis + llamada al enpoint HTTP por medio de lambda = **0,02 USD/mes**
Google Cloud Scheduler	3 tareas gratis + 9 tareas * 0,10USD/tarea = **0,9 USD/mes**
Microsoft Azure	2150 * 0,000125 USD/tarea = **0,26875 USD/mes**

Caso de Uso 2: En este caso, el estudio es de una empresa de menor tamaño que tiene pocos servicios que se deben ejecutar de manera asíncrona. Los procesos son los siguientes.

De nuevo vuelve a destacar AWS-EB para esta tarea, posicionándose como primero el uso de AWS-EB programador.

Cuadro 3.5: Servicios Caso 2

ID	**Servicio**	**Frecuencia**	**Número de llamadas al mes**
1	Sincronizar base de datos 1	3 veces al día	90
2	Realizar proceso de limpieza	1 vez a la semana	4
3	Copia de seguridad	12 veces al día	360
4	Descargar archivos de internet	1 vez al día	30
		Total:	**484**

Cuadro 3.6: Precio por servicio Caso 2

Servicio	**Precio**
AWS-EB reglas	484 * 1/1.000.000 USD por tarea + llamada al enpoint HTTP (484 * (0,24/1.000.000 USD)) = 0,000484 + 0,00011616 = **0,00060016 USD/mes**
AWS-EB programador	14 millones de invocaciones gratis + llamada al enpoint HTTP por medio de lambda = **0,00 USD/mes**
Google Cloud Scheduler	3 tareas gratis + 1 tarea * 0,10USD/tarea = **0,1 USD/mes**
Microsoft Azure	484 * 0,000125 USD/tarea = **0,0605USD/mes**

3.3 Microservicios

Definición

Las aplicaciones desarrolladas en los últimos tiempos tienden a cambiar de manera dinámica muy frecuentemente en el tiempo [RNM06], debido a cambios en la naturaleza del negocio, nuevas tecnologías que deben emplearse en la aplicación o nuevos requisitos del cliente. Por ello, se necesita una arquitectura que favorezca el desarrollo de estos cambios de manera que una aplicación sea escalable y se permita añadir nuevos servicios y modificar los existentes sin alterar el resto de componentes de la aplicación, ahorrando costes y tiempo en el desarrollo.

Los microservicios son un enfoque arquitectónico para el desarrollo de software [Ric18]. No existe una definición precisa para este termino, sin embargo, se relaciona en torno a la idea de desarrollar aplicaciones complejas divididas en pequeños servicios.

Según Sam Newman [New15]:

> "Los microservicios son servicios pequeños y autónomos que trabajan juntos"

Martin Fowler describe a los microservicios como[12]:

> "Conjuntos de servicios independientes y desplegables"

En definitiva, se puede decir que un microservicio trata de una unidad independiente que forma parte de una aplicación de mayor tamaño y que se comunica con el resto de unidades mediante mensajes (figura 3.7). El objetivo de los microservicios es facilitar el desarrollo y mantenimiento de una aplicación dividiéndola en pequeños módulos que trabajan juntos. De esta forma, la expansión o cambios en la aplicación se puede realizar modificando el módulo correspondiente.

Este término data del 2005, cuando Peter Rodgers introdujo el término "Micro-Servicios-Web" durante su presentación en la conferencia Web Services Edge [WDC+21]. Rodgers trató de aplicar la filosofía de Unix en su desarrollo, es decir, escribir programas que realicen una única función y la realicen bien. Este enfoque cumple con el "Principio de Responsabilidad Única" del Clean Code [Mar08], una filosofía del desarrollo de software para facilitar la escritura y lectura de código.

[12] `https://martinfowler.com/articles/microservices.html`

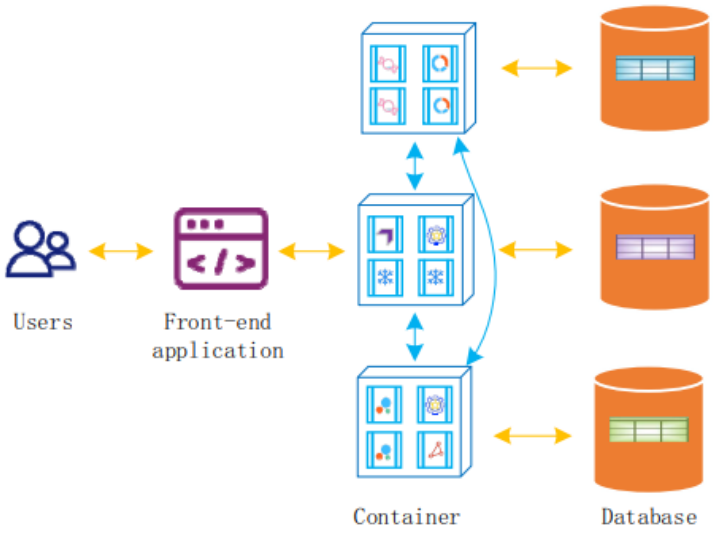

Figura 3.7: Arquitectura de microservicios. Imagen obtenida de: [LHL+20]

Los microservicios, a pesar de sus ventajas, introducen una nueva capa de complejidad en la arquitectura de software. Estos componentes independientes, aunque pueden considerarse como partes separadas de una misma aplicación, requieren establecer un mecanismo de comunicación entre sí. Esta comunicación se vuelve esencial en situaciones donde distintos microservicios necesitan cooperar para llevar a cabo tareas conjuntas. Por ejemplo, en el contexto de una tienda en línea, durante el proceso de compra, es necesario que el microservicio encargado de la autenticación del usuario interactúe con el microservicio responsable de gestionar las transacciones para garantizar la veracidad y seguridad de la transacción.

La comunicación entre los microservicios (figura 3.8) puede ser síncrona (se envía un mensaje y se bloquea el hilo de ejecución hasta que se reciba la respuesta) o puede ser asíncrona (se envía un mensaje sin esperar la respuesta).

En las comunicaciones síncronas, es común utilizar Representational State Transfer (REST). REST es un estilo arquitectónico para diseñar servicios web que utiliza operaciones HTTP estándar (GET, POST, PUT, DELETE) para manipular recursos.

Por otro lado, en las comunicaciones asíncronas, RabbitMQ y Apache Kafka son tecnologías con bastante uso. RabbitMQ es un sistema de mensajería de código abierto que implementa el protocolo Advanced Message Queuing Protocol (AMQP). Kafka es una plataforma de transmisión de datos distribuida que se utiliza para la construcción de sistemas de mensajería y procesamiento de datos en tiempo real.

Los microservicios también pueden comunicarse mediante middlewares (intermediario entre los microservicios), aunque, en estos casos, el middleware se utiliza para habilitar la interacción entre el cliente y el servicio. Los servicios exponen su funcionalidad mediante APIs. Las API REST o Apache Thrift API son comúnmente utilizados.

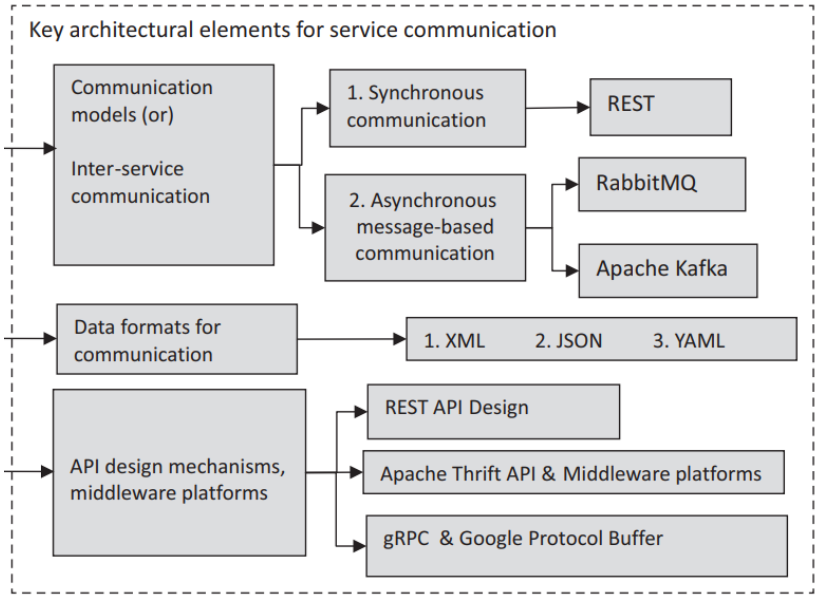

Figura 3.8: Tipos de comunicación en los microservicios. Imagen obtenida de: [SGP19]

Otros tipos de arquitectura

Arquitectura monolítica La arquitectura monolítica es un método tradicional para el desarrollo de software [DL19]. En este tipo de arquitecturas, las distintas funcionalidades se combinan en un único programa para dar lugar a una aplicación (figura 3.9). Es especialmente útil para aplicaciones de menor envergadura, ya que simplifica el proceso de desarrollo y despliegue al consolidar todas las funcionalidades en un solo programa [JVS21].

Sin embargo, el mantenimiento de la aplicación es muy costoso al tener una gran cantidad de código y la expansión o actualización de las funcionalidades de la aplicación obligan a la reestructuración completa de la aplicación.

En comparación a la arquitectura de microservicios, la arquitectura monolítica es más fácil de desarrollar y de desplegar, pero más compleja de mantener, es más propensa a errores y más difícil de escalar [GZ20].

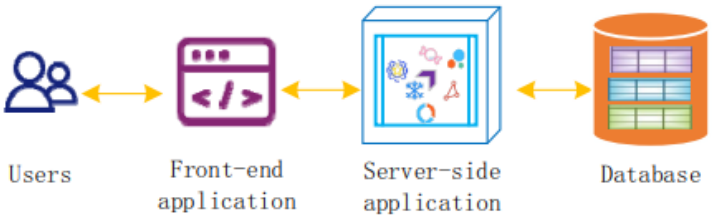

Figura 3.9: Arquitectura monolítica. Imagen obtenida de: [LHL+20]

Service-oriented architecture (SOA) Las SOA es un método de desarrollo de software que utiliza componentes llamados servicios para crear aplicaciones empresariales [LL09]. Cada servicio ejecuta una función específica, por ejemplo, un servicio que sirve únicamente para registrarse. Estos servicios poseen toda la funcionalidad para realizar su tarea, desde el propio servicio, la interfaz y la comunicación con el resto de servicios (figura 3.10).

En comparación a la arquitectura monolítica, entre otras ventajas, permiten agregar nuevas funcionalidades de manera flexible y es más tolerante a errores.

Tienen muchos puntos en común con la arquitectura de microservicios, sin embargo, su principal diferencia es el alcance, las SOA tienen un alcance empresarial y los microservicios un alcance de aplicación [Ric15].

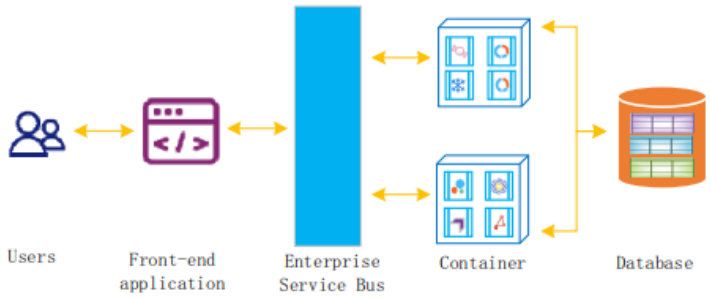

Figura 3.10: Arquitectura orientada a servicios. Imagen obtenida de: [LHL+20]

Contenedores

Según Docker, una de las empresas líderes en el desarrollo de los contenedores: "Un contenedor es una unidad estándar de software que empaqueta el código y todas sus dependencias para que la aplicación se ejecute de manera rápida y confiable de un entorno informático a otro"[13].

Son una alternativa a las máquinas virtuales aunque mantienen ciertas propiedades como el aislamiento y la asignación de recursos (figura 3.11). A diferencia de las máquinas virtuales explicadas en la sección 3.2.1, no utilizan un hipervisor si no que comparten el kernel del sistema operativo, es decir, virtualizan el sistema operativo en lugar del hardware, por lo que son más simples y rápidas de crear y de eliminar [And15].

[13] `https://www.docker.com/resources/what-container/`

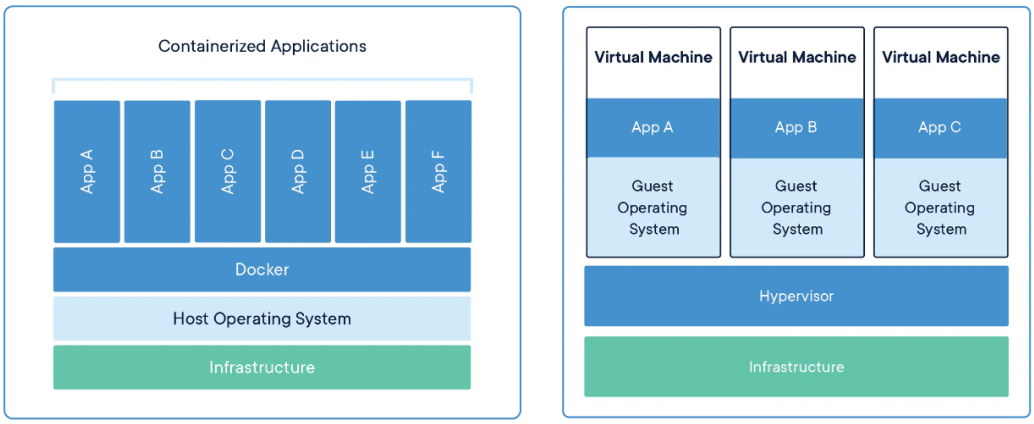

Figura 3.11: Comparación máquina virtual vs contenedor. Imagen obtenida de: Docker

En el contexto de los microservicios, los contenedores son la herramienta utilizada para encapsular y desplegar los diferentes servicios. Estos contenedores se han convertido en un estándar dentro de la industria del desarrollo de software [KMA19], herramientas como AWS o Google Cloud ofrecen servicios para la administración de contenedores y su despliegue en un sistema de computación en la nube. Los contenedores son ideales para encapsular microservicios ya que sus ventajas coinciden con los requisitos deseados por los microservicios [NMMA16]

- *Agilidad*: el despliegue de servicios basados en contenedores es una práctica que acelera el desarrollo y el despliegue de las aplicaciones.
- *Escalabilidad*: los contenedores se pueden escalar para abordar las distintas cargas de trabajo, tanto si se necesitan más recursos como si se quiere reducir el consumo.
- *Portabilidad*: otra de las ventajas de los contenedores es que se pueden ejecutar en diferentes entornos, ya que el contenedor encapsula las dependencias para que no sea dependiente del sistema operativo o dispositivo en el que se ejecuta.
- *Aislamiento*: los contenedores se ejecutan como procesos independientes con su propio sistema en el cual los elementos externos no interfieren dentro del contenedor. De hecho, los contenedores también son independientes unos de otros.

3.4 Decision Support System (DSS)

Un sistema de soporte a la decisión DSS se define como un sistema de información interactivo diseñado para ayudar a resolver problemas de decisión. También se puede definir como un sistema de cómputo que representa y procesa el conocimiento de forma que la toma de decisiones sea más productiva, ágil, innovadora y que satisface a los stakeholders de un proyecto. [BH08].

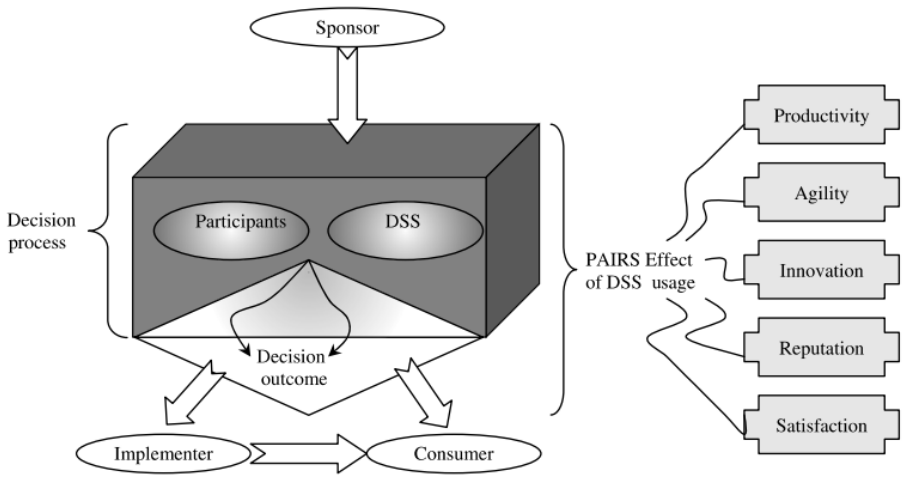

Figura 3.12: El rol de un DSS en el proceso de decisión. Imagen obtenida de [BH08]

Además de ser un almacén de conocimiento, el soporte que ofrece un DSS puede incluir la extracción de conocimiento de otras fuentes, la creación de nuevo conocimiento o la presentación de la información en distintos formatos.

El proceso de la toma de decisiones propuesto por Simon [Sim60] consta de tres fases: inteligencia, diseño y elección. En la fase de inteligencia, se busca el problema, en la fase de diseño, se consideran alternativas, por último, en la fase de elección, se decide qué alternativa elegir. En la figura 3.12, la caja negra representa este proceso de decisión, teniendo en cuenta tanto a los participantes como al propio DSS durante la decisión.

La importancia del conocimiento y de los sistemas de decisión ha sido investigada durante tiempo pero no fue hasta finales de 1980 que empezaron a surgir los términos conocimiento y sistemas de decisión en fuentes científicas [BHW81].

Los DSS se convirtieron en una herramienta popular dentro de los sistemas de información durante los siguientes años, coincidiendo con la expansión de la gestión del conocimiento. Desde 1988 en adelante, la cantidad de artículos científicos que tratan este tema creció de manera exponencial, llegando a su pico máximo durante los años 2003 al 2005 como muestra la figura 3.13.

Arquitectura

Un sistema de soporte a la decisión tiene cuatro componentes principalmente [BH08]:

- Sistema de Lenguaje: los mensajes que un DSS entiende.
- Sistema de Presentación: los mensajes que un DSS emite.
- Sistema de Conocimiento: el conocimiento que un DSS almacena.
- Sistema de Procesamiento del Problema: es el elemento que permite la conexión de los anteriores sistemas, funciona como un motor de software para un DSS.

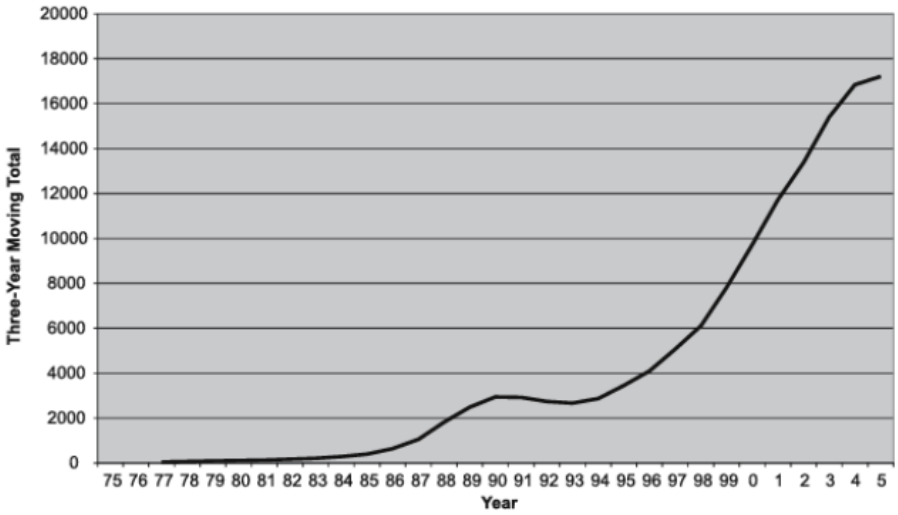

Figura 3.13: Número de artículos científicos sobre DSS o Conocimiento. Imagen obtenida de [BH08]

El procesador de problemas es el elemento que reconoce y resuelve los problemas. En la figura 3.14 se puede ver la integración de los cuatro elementos de un DSS. El usuario interacciona con el Sistema de Lenguaje, seleccionando aquella opción que necesite, por ejemplo puede añadir nuevo conocimiento o pedir ayuda. El sistema de lenguaje contacta con el Sistema de Procesamiento, el cual selecciona el conocimiento que crea necesario e incluso es capaz de crear nuevo conocimiento o modificarlo si es necesario debido a la naturaleza del problema. Por último, el sistema de procesamiento emite una salida al usuario con el Sistema de Presentación.

Esta no es la única forma de activar un DSS; este sistema también se puede activar en base a eventos. Si llegara nuevo conocimiento que implicara cambios en alguna decisión anterior, el DSS es capaz de emitir alertas al usuario.

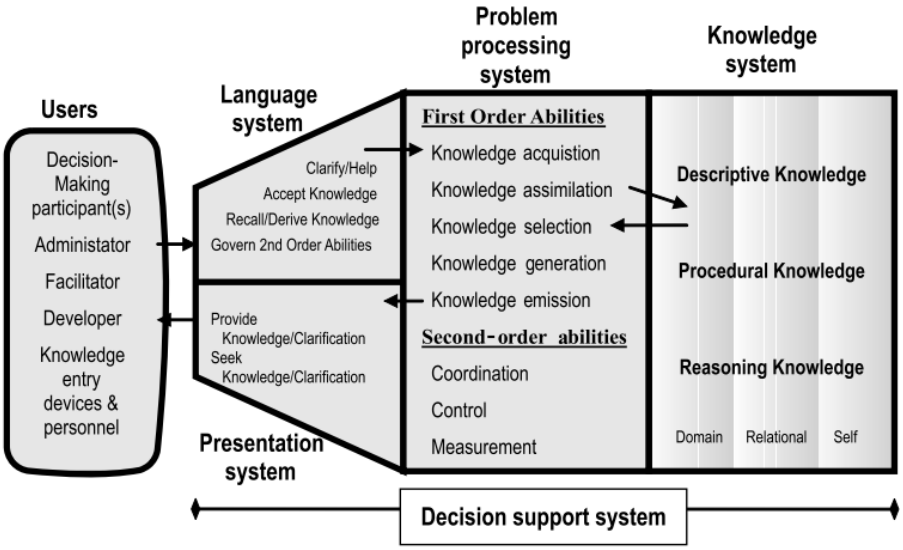

Figura 3.14: Arquitectura de un DSS. Imagen obtenida de [BH08]

En el Sistema de Procesamiento encontramos dos secciones, habilidades de primer orden y de segundo orden.

Las habilidades de primer orden comprenden las principales actividades de manipulación del conocimiento descrito identificadas en [Hol03]. Estas habilidades permiten al DSS tomar una decisión en un episodio mediante una serie de pasos lógicos.

Las habilidades de segundo orden supervisan el buen funcionamiento de las habilidades de primer orden. Primero, la coordinación permite organizar las tareas de la manipulación del conocimiento en secuencias de interés para llegar al resultado deseado. El control garantiza la calidad, validez y seguridad durante el transcurso de un proceso de decisión. La medición es la capacidad de seguir el procesamiento y los resultados dentro y entre episodios de toma de decisiones en términos de criterios deseados.

La arquitectura de un DSS reconoce que varios tipos de conocimiento que son alojados dentro del Sistema de Conocimiento. Los más básicos son el conocimiento descriptivo, el conocimiento procedimental y el conocimiento de razonamiento. El primero es conocimiento que describe el estado de algún mundo de interés. El conocimiento procedimental caracteriza cómo hacer algo, es una especificación paso a paso de qué hacer para lograr alguna tarea. El conocimiento de razonamiento especifica qué conclusión es válida cuando se sabe que existe una situación específica, especifica la lógica que vincula una premisa con una conclusión.

Tipos de DSS

Los DSS tuvieron que subdividirse en diversos tipos para cumplir las necesidades de cada tipo de usuario, incluyendo operaciones, gestión financiera u optimización.

Los Group Decision Support System (GDSS) por una parte, se encargan de facilitar la solución para problemas no estructurados o semi estructurados por un grupo de decisiones que trabajan conjuntamente como equipo, el brain storming o la evaluación de ideas son algunas de las aplicaciones [Bid96].

Por otro lado, el Enterprise Information System (EIS) se centra en pequeños grupos de trabajo, principalmente cargos ejecutivos encargados de tomar decisiones estratégicas en una organización [LE93]. Presenta el estado de un negocio mediante informes para facilitar la monitorización de la empresa. Es el precursor del Business Intelligence (BI), la principal diferencia es la potencia del BI, el cual en comparación al EIS, amplia y enriquece sus funcionalidades permitiendo integración de distintas fuentes o presentaciones más interactivas.

Knowledge-Driven Decision Support System (KD-DSS) es otra especialización de un DSS, en este caso se emplea únicamente conocimiento experto sobre un dominio particular [BH08]. Es un término muy relacionado con el Data Mining. En comparación a otros DSS los cuales tratan de simular un procesamiento de razonamiento humano, un KD-DSS trata de resolver un problema en base a su conocimiento y reglas lógicas.

3.5 Large Language Model (LLM)

Definición

Los modelos fundacionales son un tipo de modelos de IA entrenados con una gran cantidad de datos de distintos tipos, desde texto e imágenes hasta código de programación [BHA+21]. Estos modelos pueden hacer una gran cantidad de tareas con mucha precisión como la generación de contenidos (imágenes, sonidos o texto), el procesamiento de lenguaje natural NLP o la clasificación de textos.

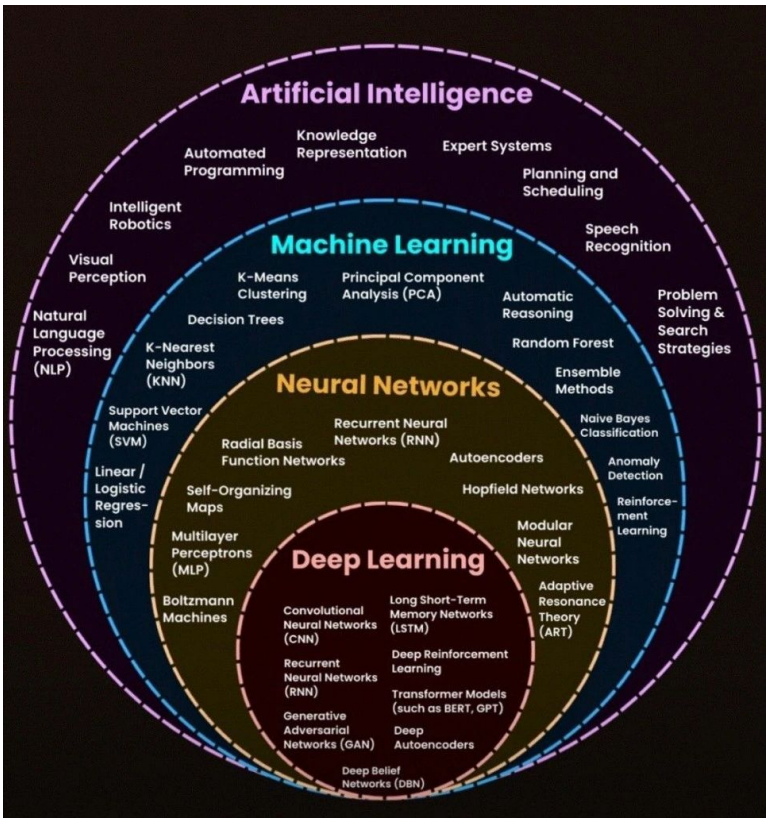

Figura 3.15: Diagrama de Venn de la IA. Imagen obtenida de Mohini

Los LLM son una instancia de los modelos fundacionales diseñados para procesar y generar lenguaje humano [Ama23]. A diferencia de los modelos fundacionales, estos tienen un objetivo mucho más específico, relacionado únicamente con el lenguaje, más adelante se detallarán sus aplicaciones. Los LLM son modelos con un mayor desarrollo y madurez que los modelos fundacionales y además su uso está más extendido. Tanto los modelos fundacionales como los LLM se encuadran como modelos de IA. Si bien es cierto que la IA abarca muchos campos, los LLM se consideran modelos de Deep Learning (figura 3.15).

Estos modelos se les denomina como Large (Largos) por los siguientes motivos:

- Los LLM son modelos que se entrenan con una gran cantidad de datos, llegando a los petabytes. Los LLM se entrenan con textos extraídos de diversas fuentes como artículos científicos, foros como Stack Overflow[14] o la Wikipedia[15].
- También se les considera largos por la cantidad de parámetros con los que funciona. Los parámetros en un LLM indica un valor dentro de la red neuronal que explica como el modelo genera texto. Los parámetros indican la relación entre las palabras y frases dentro de los datos con los que se ha entrenado el modelo. Estos parámetros pueden ser los pesos (valores que se multiplican por las entradas de cada neurona) o sesgos (valores que se suman a la salida de cada neurona). En muchos casos, se indican los parámetros de los LLM, en el caso de GPT-3, cuenta con 175B (billones) de parámetros [KS22].

La base para explicar como funciona un LLM son los transformadores. Un transformador es un modelo específico de red neuronal, publicado por Google en 2017 [VSP+23]. Los transformadores evolucionaron desde las redes neuronales recurrentes, permitiendo el procesamiento de secuencias de entrada en paralelo, mejorando su eficiencia durante el entrenamiento, por lo que en el mismo tiempo son capaces de entrenar con muchos más datos.

Los transformadores son capaces de generar la siguiente palabra dado un texto con el siguiente procedimiento. Primero, toman el texto y lo dividen en partes (tokens), en el caso del lenguaje, son palabras.Estos tokens se modifican para que sean interpretables por un computador, este proceso se le llama embedding y el resultado es un vector de números reales, capturando el significado semántico y las relaciones lingüísticas de la palabra.

A continuación, el componente de atención de un transformador, se considera el componente más importante del transformador, averigua que tokens son importantes para conocer el contexto. Para ello, calcula el peso de cada par de tokens en un texto y este peso determina la atención que se le debe prestar a otras palabras en una secuencia.

Después, los resultados de los vectores se pasan a un perceptrón multicapa (figura 3.16), un tipo de red neuronal compleja que consta de varias capas de nodos. Este perceptrón multicapa constra de una capa de entrada, una serie de capas ocultas y una capa de salida. Su objetivo es refinar los vectores obtenidos por el componente de atención. Una vez termina el perceptrón multicapa, se le vuelve a enviar los resultados al componente de atención y este proceso se repite durante un número de veces.

Finalmente, se obtiene una distribución de probabilidad con los vectores que representan los posibles siguientes tokens como resultado. Para obtener estos tokens hay que deshacer el vector y obtener el token mediante el unembedding.

[14] `https://stackoverflow.com/`
[15] `https://es.wikipedia.org/wiki/Wikipedia:Portada`

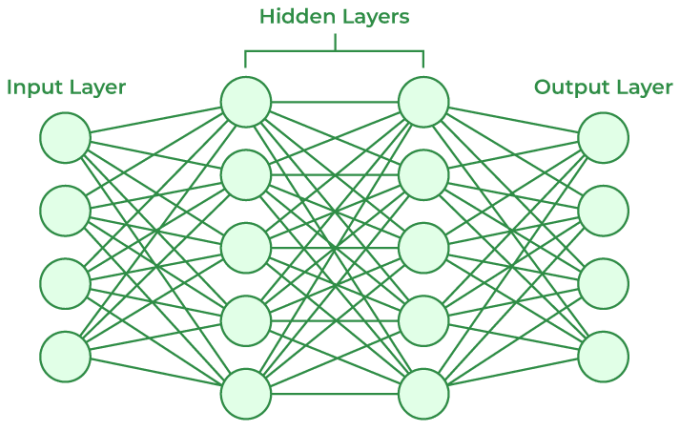

Figura 3.16: Representación de un perceptrón multicapa. Imagen obtenida de GeeksforGeeks

Aplicaciones

Los LLM han irrumpido con fuerza en los últimos años y ha sido gracias a las aplicaciones que ofrece. Algunas de sus aplicaciones son las siguientes:

- **Generación de texto:** es una de las vías más investigadas por las empresas. Los LLM pueden generar diversos tipos de textos formales, como informes o artículos científicos. Estos modelos son capaces de adaptar el estilo y el tono a las necesidades específicas del usuario.
- **Clasificación de texto:** los LLM pueden clasificar automáticamente documentos en función de su contenido, temática o tipología. Esta capacidad es especialmente útil para organizar grandes volúmenes de información. Existen varias aplicaciones que proporcionan un gran rendimiento para la clasificación de textos jurídicos [CGB+21].
- **Resumen de textos:** los LLM pueden identificar y extraer los puntos más importantes de un texto extenso, creando un resumen conciso y preciso. Esta función es útil para ahorrar tiempo en la lectura de documentos largos
- **Chatbot:** es la aplicación por excelencia de los LLM, desde la irrupción de ChatGPT[16], esta aplicación ha aumentado el interés por la IA y en concreto por los LLM. Un chatbot es un programa de software que simula una conversación con usuarios humanos, generalmente a través de interfaces de texto.

[16] https://chatgpt.com/

Capítulo 4

Metodología

La metodología empleada en la realización del proyecto se detalla en el siguiente capítulo. En primer lugar, se aborda la metodología de desarrollo utilizada. Posteriormente, se presenta la planificación del trabajo realizada. Por último, se proporciona información sobre los recursos de hardware y software utilizados en la ejecución del proyecto.

4.1 Metodología de Desarrollo

Para asegurar el éxito en el desarrollo de este proyecto, se necesita seguir una metodología de desarrollo que asegure la obtención de un producto final de calidad. Este proyecto requiere una metodología que sea flexible con los requisitos y que permita modificarlos durante el desarrollo ya que se tiene cierta incertidumbre en cuanto a los requisitos del proyecto. En este contexto, las metodologías ágiles representan un gran valor añadido para un proyecto [SW07].

Las metodologías ágiles, surgidas en 2001 a través del "Manifiesto por el Desarrollo Ágil de Software"[1], se centran en un enfoque flexible y colaborativo que permite adaptarse rápidamente a los cambios. Estas metodologías tienen como objetivo principal entregar valor al cliente de manera más rápida y eficiente. Algunos de los beneficios que pueden proporcionar a las empresas incluyen una mayor capacidad de respuesta a las necesidades del cliente, una mejor gestión del riesgo, una mayor calidad del producto final y una mejora en la satisfacción del cliente. Además, fomentan la colaboración entre los miembros del equipo y promueven un enfoque más iterativo y experimental en el desarrollo de software.

El manifiesto ágil se basa en los siguientes cuatro valores del desarrollo ágil:

- **Individuos e interacciones** sobre procesos y herramientas
- **Software funcionando** sobre documentación extensiva
- **Colaboración con el cliente** sobre negociación contractual
- **Respuesta ante el cambio** sobre seguir un plan

Priorizando los elementos de la izquierda sobre los de la derecha, aunque estos siguen siendo importantes.

[1] `https://agilemanifesto.org/iso/es/manifesto.html`

En comparación, las metodologías tradicionales, tales como la metodología en cascada o el modelo en espiral tienden a seguir un enfoque más rígido y secuencial, centrado en la planificación detallada y la documentación exhaustiva de requisitos [Awa05]. Dada la naturaleza de este proyecto se ha elegido una metodología ágil que se amolda a los requerimientos, dento de este tipo de metodologías, se ha elegido Scrum para el desarrollo.

4.2 Scrum

Scrum es un marco de trabajo ágil basado en un enfoque iterativo e incremental, dividido en ciclos cortos de trabajo llamados sprints [Rub12]. Durante cada sprint, el equipo se enfoca en entregar un conjunto de funcionalidades prioritarias, revisando y adaptando su planificación en retrospectivas regulares al final de cada ciclo (figura 4.1.

Scrum fomenta la colaboración estrecha entre los miembros del equipo así como la entrega rápida y frecuente de valor al cliente. Las reuniones son el pilar fundamental de la metodología, donde diferenciamos entre reuniones de planificación, diaria, de revisión y de retrospectiva, la más importante de todas ellas, ya que, se realiza después de terminar un sprint para reflexionar y proponer mejoras en los avances del proyecto.

Este marco de trabajo se apoya en roles definidos, como el Scrum Master, el Product Owner y el equipo de desarrollo, así como en artefactos como el Product Backlog y el Sprint Backlog, que ayudan a organizar y priorizar el trabajo.

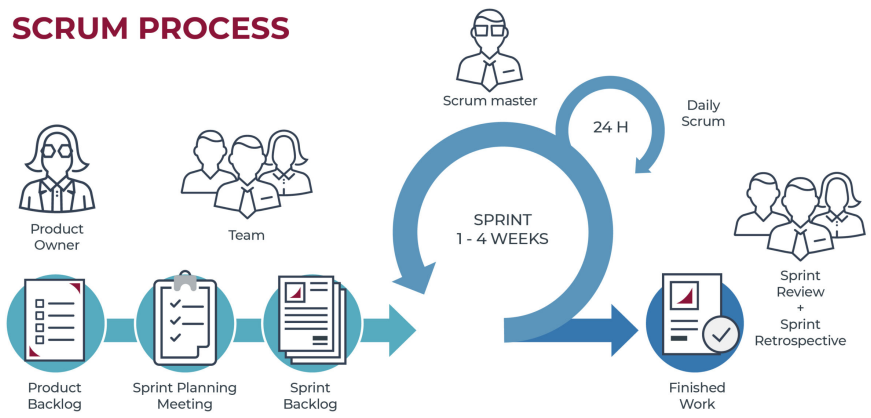

Figura 4.1: Proceso Scrum. Imagen obtenida de UEMC

Se han modificado ciertos aspectos de Scrum para adaptarlo al desarrollo de este proyecto. En primer lugar, el Scrum Master y el Product Owner será la misma persona, en este caso el tutor de NTT Data. En segundo lugar, las reuniones no serán diarias, ya que el equipo de desarrollo es únicamente el autor de este TFG, si no que serán dos semanales para garantizar el correcto progreso del proyecto.

4.3 Planificación de Trabajo

En esta sección se define la planificación del trabajo para la consecución del proyecto. Utilizando Scrum como marco de trabajo, se ha dividido la carga de trabajo en Sprints. El número de sprints y su duración se justifica por la estructura incremental y adaptable de Scrum. Con siete sprints de una a cuatro semanas, se logra un equilibrio entre avances continuos y flexibilidad para adaptarse a cambios. A continuación, se muestra un diagrama de Gantt (figura 4.2) que ilustra la división temporal a lo largo de los meses para el desarrollo de este trabajo.

Sprint 1 **Revisión bibliográfica (19/02-11/03)**. En este sprint se estudiará el estado del arte respecto a las tecnologías utilizadas en el proyecto. Se ha estudiado el estado del arte respecto a las plataformas de computación en la nube y la programación de tareas.

Sprint 2 **Definición de la arquitectura (12/03-01/04)**. Una vez conocidas las tecnologías principales para el desarrollo del proyecto, se ha realizado una serie de entrevistas para la extracción de los requisitos y el diseño de la arquitectura del proyecto.

Sprint 3 **Planificación de la implementación (02/04-08/04)**. Definición de la planificación de la implementación, por un lado se definirán las fechas de entregas y se seleccionarán las tecnologías y frameworks con los que se trabajará.

Sprint 4 **Implementación de microservicios backend (09/04-29/04)**. A partir de este sprint comienza el desarrollo de la aplicación, se implementarán los microservicios correspondientes al backend, en la sección 5 se detallarán estos microservicios.

Sprint 5 **Implementación de frontend e integración (30/04-20/05)**. Se desarrollará el frontend de la aplicación y además, se orquestarán los microservicios con Kubernetes

Sprint 6 **Pruebas y depuración (21/05-03/06)**. Se realizarán pruebas sobre los servicios, centrándonos en el correcto funcionamiento de la aplicación y se realizarán informes para corroborar estas pruebas. Además, se depurará la aplicación mejorando la eficiencia.

Sprint 7 **Implementación de mejoras (04/06-17/06)**. Los informes del Sprint 6 se evaluarán y se solucionarán los errores pertinentes además de implementar las mejoras propuestas por el cliente para la consecución del trabajo.

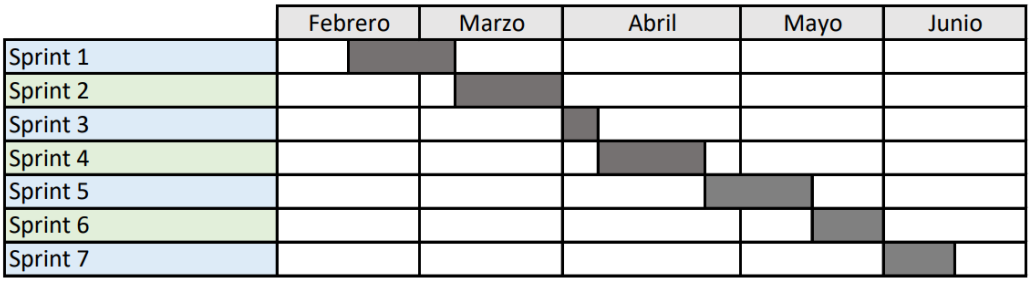

Figura 4.2: Diagrama de Gantt que muestra la planificación del proyecto

4.4 Recursos

En esta sección se describen los recursos empleados para este proyecto. Se divide entre recursos de hardware y recursos de software.

4.4.1 Recursos Hardware

En esta sección se describen los recursos de hardware utilizados. Se ha utilizado un ordenador para el desarrollo de este proyecto. Este ordenador ha sido brindado por NTT Data, la empresa con la que se colabora para el desarrollo de este TFG. Además, se ha utilizado un ordenador personal para el despliegue de la aplicación con Ubuntu.

4.4.2 Recursos Software

A continuación se describen los recursos de software utilizados durante el desarrollo del proyecto. Se clasifica cada recurso en distintas secciones para facilitar la comprensión de cada elemento.

Sistema Operativo

- **Windows 10**[2]. Fue el sistema operativo principal durante el desarrollo de la aplicación. Se utilizó debido a su facilidad de uso y conocimiento de las herramientas.
- **Ubuntu 22.04**[3]. Ubuntu se utilizó para desplegar los microservicios con Docker y Kubernetes ya que ofrece mayores posibilidades al tener un soporte nativo en dicho sistema operativo.

Lenguajes de Programación

- **JAVA**[4]. JAVA ha sido empleado para gran parte del desarrollo del backend, siendo creadas las APIs. Este lenguaje es ampliamente utilizado para el desarrollo del backend de aplicaciones.
- **Dart**[5]. Dart es el lenguaje utilizado para el desarrollo del frontend de la aplicación. Se utilizó por el conocimiento previo de la aplicación y el gran soporte que tiene al ser propiedad de Google.
- **Python**[6]. Python ha sido utilizado para el desarrollo del modelo de lenguaje. Este lenguaje es uno de los más utilizados en el entrenamiento de modelos de IA.

[2] https://www.microsoft.com/es-es/windows?r=1
[3] https://ubuntu.com/desktop
[4] https://www.java.com/es/
[5] https://dart.dev/
[6] https://www.python.org/

Frameworks

- **Flutter**[7]. Flutter es un framework de código abierto creado por Google para el desarrollo de aplicaciones multiplataforma. Se utilizó para integrar el backend y el frontend de la aplicación.
- **Spring**[8]. Spring es el framework para el desarrollo de JAVA más popular. Se utilizó para la creación de los microservicios de manera rápida y sencilla.
- **FastAPI**[9]. FastAPI es uno de los frameworks más populares para el desarrollo rápido de APIs web con Python. Se utilizó para implementar la API destinada al modelo de lenguaje.

Herramientas de Desarrollo

- **Docker**[10]. Docker es una plataforma de código abierto diseñada para desarrollar, enviar y ejecutar aplicaciones en contenedores. Los contenedores son unidades de software que empaquetan el código y todas sus dependencias. Se ha utilizado para encapsular los microservicios.
- **K3s - Lightweight Kubernetes**[11]. K3s es una distribución de Kubernetes ligera y fácil de instalar. Kubernetes es un sistema de código abierto para automatizar el despliegue, el escalado y la gestión de aplicaciones en contenedores. Se ha utilizado para orquestar los contenedores.
- **Git**[12]. Git es un sistema de control de versiones de código abierto. Su propósito es llevar registro de los cambios en archivos y coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.
- **GitHub**[13]. GitHub es una plataforma en línea de desarrollo colaborativo que utiliza el sistema de control de versiones Git. Es la plataforma donde se almacena el proyecto.
- **Visual Studio Code**[14]. Visual Studio Code es un IDE desarrollado por Microsoft utilizado para todos los desarrollos. Desde el frontend, los microservicios con Spring y el entrenamiento del modelo de lenguaje.
- **PostgreSQL**[15]. PostgreSQL, también conocido como Postgres, es un sistema de gestión de bases de datos relacional de código abierto y potente. Es el sistema de base de datos elegido para el almacenamiento persistente de la aplicación.

[7] `https://flutter.dev/`
[8] `https://spring.io/`
[9] `https://fastapi.tiangolo.com/`
[10] `https://www.docker.com/`
[11] `https://k3s.io/`
[12] `https://git-scm.com/`
[13] `https://github.com/`
[14] `https://code.visualstudio.com/`
[15] `https://www.postgresql.org/`

- **CockroachDB**[16] es una base de datos relacional distribuida de código abierto, diseñada para ofrecer escalabilidad, consistencia y resiliencia en entornos de producción.
- **Nginx**[17]. Nginx es un servidor web de código abierto, de alto rendimiento y escalabilidad, así como un servidor proxy inverso y un balanceador de carga. La página web se desplegará con este servidor.
- **Ngrok**[18]. Ngrok es una herramienta de código abierto que proporciona acceso seguro a los recursos locales de una red a través de internet. Permite crear túneles seguros desde una máquina local hasta un servidor público en la nube. Se ha utilizado para permitir el acceso a la página web desde Internet.
- **AWS**[19]. AWS es una plataforma de servicios en la nube líder a nivel mundial. Ofrece una amplia gama de servicios de computación en la nube. En el proyecto se ha utilizado una serie de servicios que ofrece como es la programación de tareas con AWS-EB o las funciones Lambda.
- **Jira**[20]. Jira es una herramienta de gestión de proyectos desarrollada por Atlassian, se ha utilizado para la gestión del proyecto, definiendo el Product Backlog o el Sprint Backlog.

Herramientas de Documentación

- **Overleaf**[21]. Overleaf es una plataforma en línea que ofrece un entorno de edición colaborativo para la creación de documentos LaTeX. Se ha utilizado para el desarrollo de la documentación del proyecto.
- **Draw.io**[22]. Draw.io es una herramienta en línea gratuita para crear diagramas y diagramas de forma rápida y sencilla. Se ha utilizado para crear los diagramas que aparecen en esta documentación debido a su facilidad de uso.

[16] https://www.cockroachlabs.com/
[17] https://www.nginx.com/
[18] https://ngrok.com/
[19] https://aws.amazon.com/es/
[20] https://www.atlassian.com/es/software/jira
[21] https://es.overleaf.com
[22] https://app.diagrams.net/

Capítulo 5

Arquitectura

ESTE capítulo explica la arquitectura de la aplicación, fundamentada sobre una serie de microservicios interconectados. Se ofrece una visión general de la arquitectura en su conjunto. Luego, se detalla cada uno de los microservicios que componen este sistema, explorando su funcionalidad y relaciones con otros componentes. A continuación, se detallan el resto de componentes necesarios para la construcción de la aplicación, como los servicios utilizados en AWS, el despliegue con K8s y la exposición de la aplicación a Internet. Por último, se analizan los patrones de diseño utilizados en el proyecto, justificando qué valor técnico aportan en la estructura y el funcionamiento de la aplicación en su conjunto.

5.1 Visión general

En este apartado se da una visión general de la aplicación (figura 5.1). En posteriores secciones se detallará cómo se ha diseñado y desarrollado cada uno de los módulos y cómo se relacionan entre sí.

La aplicación consta de una serie de microservicios independientes y con una funcionalidad específica, conectados para dar la funcionalidad de aplicación que permita programar, eliminar y modificar tareas que se ejecutan periódicamente en un sistema de computación en la nube.

Por tanto, en esta aplicación se utiliza la arquitectura de microservicios (sección 3.3), que divide la aplicación en pequeños módulos que se comunican mediante mensajes. Cada microservicio puede utilizar una tecnología específica y diferente al resto, implementando la funcionalidad diseñada de forma aislada. Comparada con otras arquitecturas, como la monolítica, los microservicios permiten una escalabilidad y un mantenimiento más sencillos, ya que es posible modificar o actualizar un microservicio sin afectar al resto de la aplicación.

A diferencia de la arquitectura en capas, que, aunque es más modular que la monolítica, todavía presenta dependencia entre capas adyacentes, la arquitectura de microservicios garantiza la independencia completa entre los módulos. Esto significa que cada microservicio puede funcionar de manera autónoma, asegurando que si uno falla, los demás continúan operativos, lo que aumenta la tolerancia a fallos y la disponibilidad del sistema.

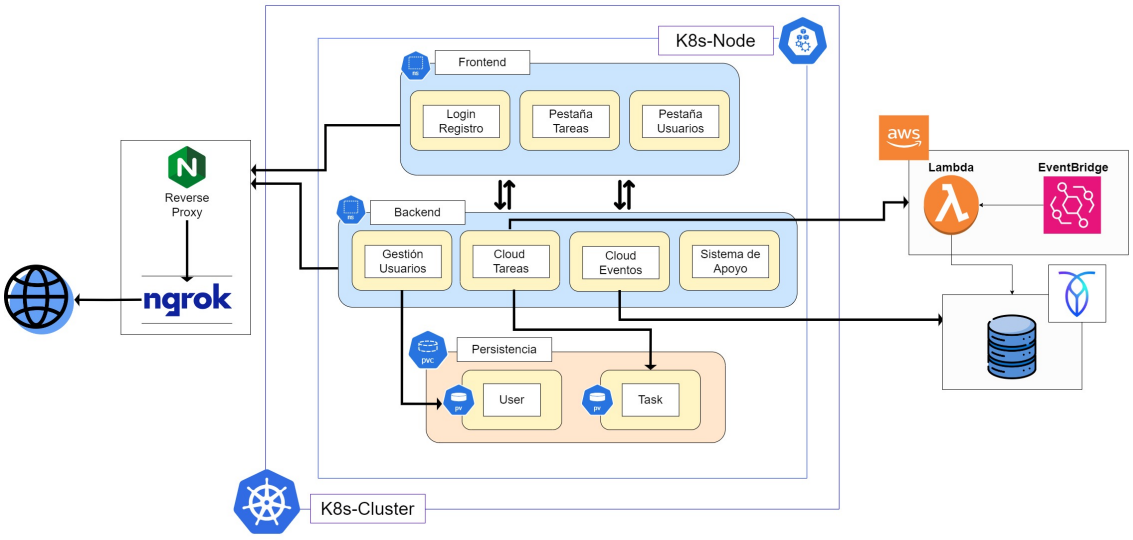

Figura 5.1: Diagrama general de la arquitectura

Otras ventajas de esta arquitectura son la significativa reducción en el tiempo de despliegue de la aplicación y la flexibilidad para implementar nuevas funcionalidades utilizando diferentes tecnologías sin comprometer la estabilidad del resto de la aplicación.

Aunque los microservicios no se organizan tradicionalmente en una arquitectura de capas, es posible representar las capas conceptuales a las que cada microservicio pertenece, facilitando una comprensión visual de su propósito y de la arquitectura en su conjunto. En este sentido, las capas conceptuales que se han definido son las siguientes:

- *Frontend.* El frontend es la parte visible de la aplicación con la que los usuarios interactúan directamente. Incluye el microservicio responsable de la capa de presentación, gestionando el diseño, la presentación visual y la interacción con el usuario.
- *Backend.* El backend es la infraestructura que soporta y gestiona la lógica de la aplicación, el almacenamiento de datos y las comunicaciones del servidor. Incluye los microservicios específicos para la gestión de usuarios, tareas en la nube (cada uno con su propia base de datos), eventos en la nube y sistemas de apoyo de la aplicación. Aunque estos microservicios integran diferentes funcionalidades del backend, cada uno opera de manera independiente, garantizando la modularidad y flexibilidad del sistema.
- *Persistencia.* La capa de persistencia es responsable del almacenamiento y la gestión de datos. Incluye las bases de datos utilizadas por los microservicios del backend (microservicio de gestión de usuarios y de tareas en la nube) para almacenar información crítica.

Estos microservicios se orquestarán con Kubernetes (K8s), que proporciona un entorno robusto y flexible para la gestión automatizada de contenedores. Esta elección se justifica porque simplifica las tareas de despliegue, escalamiento y gestión de la aplicación en un entorno de producción.

Para demostrar el funcionamiento de la aplicación, se expondrá a Internet. Todos los microservicios de la aplicación se ejecutarán localmente y se unificarán las conexiones de varios puertos en uno solo mediante un reverse proxy con Nginx. Posteriormente, se empleará Ngrok, una plataforma que permite exponer de manera segura este puerto unificado a Internet. Ngrok crea un túnel seguro desde una URL pública hacia nuestro entorno local, permitiendo el acceso a la aplicación desde cualquier parte del mundo sin comprometer la seguridad de la red local.

Por último, para programar las tareas se utilizará el servicio AWS-EB y para tener constancia de que las tareas se están ejecutando de manera correcta, se llamará a una función de lambda que escribe un registro en una base de datos de CockroachDB.

5.2 Microservicios

En esta sección se detallan cómo se han diseñado y desarrollado cada uno de los microservicios de la aplicación. Existen cinco microservicios en la aplicación, cuatro de backend y uno de frontend.

5.2.1 Gestión de Usuarios

Este microservicio, creado con Spring Boot, se encarga de administrar las cuentas de usuario en el sistema. Sus principales funciones son las siguientes.

Primero, se encarga del proceso de registro de nuevos usuarios en el sistema. Además, verifica la disponibilidad del nombre del usuario que actúa como identificador y crea un registro en la base de datos.

Segundo, se realiza la tarea de inicio de sesión. Se verifica si el nombre de usuario está registrado en el sistema y se confirma que la contraseña ingresada sea la correcta. Si todos los datos son correctos, se concede acceso al usuario.

También posibilita que los usuarios accedan a los datos almacenados en la base de datos proporcionando su nombre de usuario. Una vez verificado, el microservicio devuelve los datos del usuario, excluyendo la contraseña para evitar filtraciones.

Este microservicio gestiona el proceso de cambio de contraseña y cualquier otro dato personal que los usuarios quieran modificar. Una vez que el usuario modifica sus datos, se actualiza el registro correspondiente en la base de datos.

Por último, se ofrece a los usuarios la opción de eliminar su cuenta si así lo desean, eliminando todos los datos asociados a esa cuenta de la base de datos.

Para garantizar la seguridad de las cuentas de los usuarios, se han diseñado medidas adicionales. En primer lugar, a las contraseñas se les ha añadido una cadena de caracteres aleatorios, conocida como *salt*. A continuación, se ha cifrado la nueva contraseña en la base de datos utilizando una función hash. De este modo, no se conocerá la contraseña original. Cuando se quiera iniciar sesión, a la contraseña que introduzca el usuario se le aplicará el mismo procedimiento, se le añade un *salt* y se cifra, si esta contraseña coincide con la almacenada en la base de datos, se concederá acceso al usuario. Esta técnica aumenta significativamente la complejidad para aquellos que intenten descifrar las contraseñas, ya que ahora tendrían que descifrar tanto el hash como el *salt*.

En la figura 5.2 se puede apreciar las dos tablas que componen la base de datos de este microservicio.

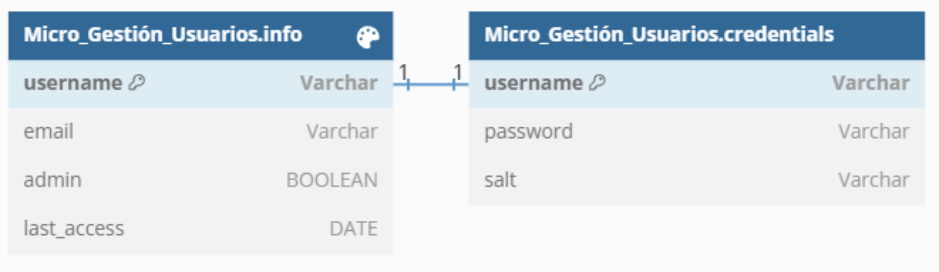

Figura 5.2: Base De Datos de los usuarios

5.2.2 Eventos Cloud

Este microservicio, creado con Spring Boot, se dedica a mostrar los resultados de las ejecuciones en AWS-EB en la base de datos de CockroachDB (figura 5.3). Los eventos que se escriben en esta base de datos se generan mediante una función Lambda de AWS (Sección 5.3). Los datos escritos en la base de datos funcionan para guardar registro de que las llamadas de AWS-EB realiza la tarea en el horario que se describa. Sus funciones principales son las siguientes.

En primer lugar, este microservicio se encarga de obtener los registros de una tarea por su identificador. Además, permite recuperar el registro completo de una tarea específica mediante su identificador único.

Por otra parte, se puede obtener el número de eventos y listar los eventos por el nombre de usuario. Permite obtener la cantidad total de eventos asociados a un usuario específico y además obtener los eventos utilizando paginación. La paginación es una consulta a través de la cual se devuelven resultados divididos en subconjuntos de datos llamados páginas, la consulta indica el tamaño de la página y se indexa a través del índice offset o desplazamiento.

Por último, se puede obtener el número de eventos y listar los eventos por el nombre de la tarea. También permite obtener el número total de eventos relacionados con una tarea específica y además listar los eventos mediante paginación para manejar más eficientemente los resultados.

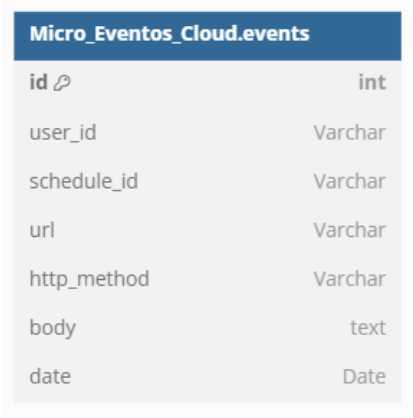

Figura 5.3: Base de Datos de los eventos generados

5.2.3 Tareas Cloud

Este es el microservicio fundamental para el desarrollo de esta aplicación y que se puede exportar a otras aplicaciones. Es el encargado de la lógica relacionada con las tareas programadas. Este microservicio se ha creado con Spring Boot y además se ha conectado con AWS mediante la dependencia *aws-java-sdk-scheduler* para la creación de las tareas en AWS-EB.

Además de almacenar las tareas en AWS-EB, los datos de la tarea se almacenan en una base de datos propia. El objetivo de la base de datos es evitar realizar múltiples llamadas al servicio en la red. Este enfoque por un lado mejorará el rendimiento al tener que esperar menos tiempo cuando se quiera obtener los datos de la tarea. Con la base de datos, se tendrá un registro exportable a cualquier otro servicio de computación en la nube si se desea cambiar la plataforma.

En la figura 5.4 se representan las tres tablas que almacenan los datos necesarios para no necesitar llamar a AWS-EB cuando se quieran listar las tareas. Además de datos que almacena AWS como el *schedule_expression* donde se indica la programación de la tarea, se almacenan otros datos de control, como el identificador del usuario al que pertenece la tarea. Como aspecto a detallar, al utilizar una única cuenta de AWS para almacenar las tareas, existe como limitación que dos tareas no pueden tener el mismo nombre. Teniendo más de un usuario que puede crear tareas, podría ocurrir el caso en el que un usuario crea una tarea y se le da un error de que el nombre ya está elegido cuando él no lo ha hecho. Por tanto, el nombre de la tarea en AWS se crea con el nombre del usuario seguido de un punto y acabando con el nombre de la tarea que le da el usuario.

Las funciones principales de este servicio son las funciones de una API que define el estándar Create, Read, Update and Delete (CRUD). Teniendo en cuenta la siguiente restricción, cuando se realizan operaciones que requieran modificación de la base de datos, se realizará tanto en la base de datos como en los registros de AWS-EB. Por otro lado, cuando se hagan consultas simplemente se accederá a la base de datos.

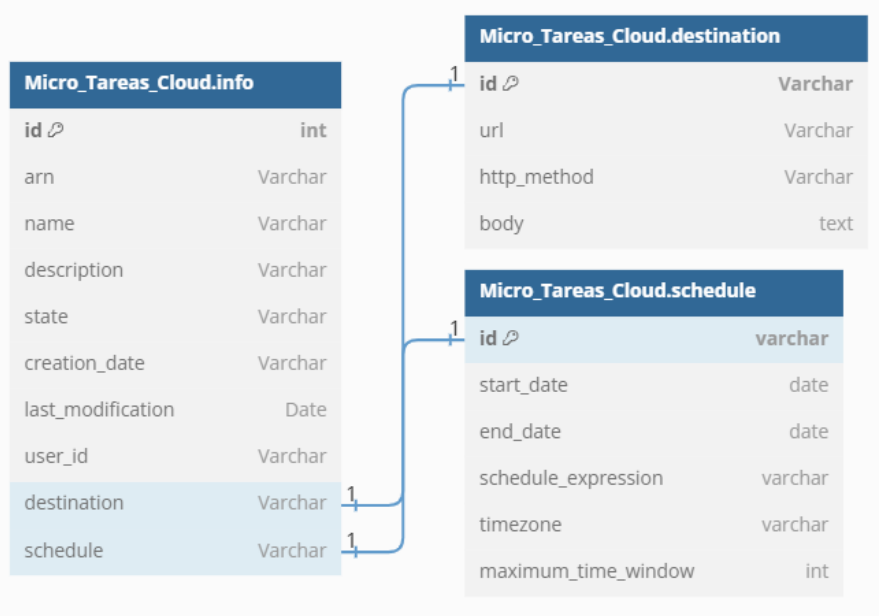

Figura 5.4: Base de Datos de las tareas

5.2.4 Sistema de Apoyo

La intención de este microservicio es facilitar la creación de tareas programadas. Por lo tanto, hay que orientar la aplicación para que usuarios no expertos puedan programar una tarea de manera sencilla. Para crear tareas con AWS se pueden utilizar diferentes especificaciones. Por un lado, se puede crear una tarea que se ejecutará una única vez. Por otro lado, se puede crear una tarea que se repetirá basándose en la frecuencia definida y por último crear una tarea con una planificación más compleja con la sintaxis de cron (Sección 3.1).

Los dos primeros casos son sencillos para el usuario, pues no tiene que declarar más que la fecha y hora en el primer caso y, en el segundo, la unidad y el valor de repetición. Sin embargo, en el caso de una expresión cron, la especificación de su sintaxis no es sencilla. Existen aplicaciones externas con las que puedes crear y definir la expresión cron. Sin embargo, se estaría forzando al cliente a tener un conocimiento previo o a utilizar herramientas externas lo que podría ser perjudicial para su satisfacción con la aplicación.

Se han barajado distintas posibilidades como sistema de apoyo. Una de ellas es la creación de la sintaxis de cron mediante la interfaz. Se rellenarían distintos campos con los que se podría personalizar la sintaxis de cron. Sin embargo, esta posibilidad podría ser demasiado compleja para el usuario al ver una gran cantidad de campos a rellenar para definir la sintaxis, resultando en una funcionalidad poco práctica.

Una segunda posibilidad es la de generar expresiones de cron que puedan ser consideradas ideales para el usuario. El planteamiento trata de crear la expresión de cron de manera automática mediante el análisis de la descripción y nombre de la tarea y de expresiones anteriores declaradas por el mismo usuario y por otros usuarios. De esa forma, el usuario se abstrae acerca de cuándo ejecutar la tarea. Para ello, se desarrollaría un sistema experto con el conocimiento de un experto en la materia creando un sistema capaz de dadas unas pautas conocer cuál será el resultado de la expresión. En este caso, el planteamiento puede fallar debido a que el usuario no tiene porque tener un patrón definido en la programación de sus tareas, necesitando así una expresión distinta para cada tarea que defina que no haya utilizado antes. Además, tampoco se resuelve el problema de la complejidad en la creación de la expresión de cron.

En este contexto, y para crear una expresión cron, se necesita una interacción lo más natural posible entre el usuario y el ordenador, por lo que otra alternativa consiste en utilizar procesamiento de lenguaje natural. Para materializar esta posibilidad se podría crear un asistente virtual que se centre en la creación de la sintaxis de cron. Este asistente será capaz, por medio de preguntas con lenguaje natural, de crear la expresión cron que quiere el usuario. En este trabajo se ha optado por abordar esta alternativa. Así, para la creación del asistente virtual, se ha utilizado un LLM. Un LLM es un modelo de aprendizaje automático capaz de comprender y generar texto en lenguaje natural y de forma que sea entendible para un humano (Sección 3.5).

En este caso no se está creando un nuevo modelo de LLM desde cero, si no que utilizando uno existente se le está entrenando para un objetivo en específico. Este enfoque, a diferencia de utilizar un LLM con un conocimiento general, es beneficioso ya que se puede mejorar los resultados obtenidos por el modelo para una tarea en concreto a expensas de perder rendimiento en otros objetivos.

Este entrenamiento se denomina formalmente como *fine-tuning* y su objetivo es ajustar las capas de la red neuronal del LLM para que este sea más preciso en el problema que se desea resolver, en este caso la definición de expresiones cron. Este fine-tuning es ideal para proyectos en los que se necesita una red neuronal pero no se cuenta con la suficiente cantidad de datos para entrenarlo desde cero ahorrando tiempo.

El fine-tuning de un LLM es un proceso complejo que requiere principalmente de dos etapas para su consecución. Primero, se debe crear un nuevo conjunto de datos con los que entrenar el modelo para que desarrolle la funcionalidad que se espera del mismo. Para ello, se debe generar a mano simulaciones de conversaciones entre un usuario y el asistente, centrado en el tipo de conversación que se espera tener. Como en este caso no se requiere un asistente conversacional si no que simplemente se necesita un asistente que sea capaz de responder a una pregunta, no es necesario que sea capaz de continuar una conversación.

Por lo tanto, en lugar de crear un dataset reproduciendo diferentes conversaciones entre usuario y asistente, se creará un conjunto de datos con el que se entrenará de nuevo al modelo con preguntas y respuestas individuales.

Las preguntas simularán cómo un usuario formularía la cuestión al modelo y la respuesta será únicamente la expresión cron generada. Además, se debe evitar que el modelo responda preguntas irrelevantes para esta cuestión, que podrían crear un desbarajuste en las respuestas para el objetivo que se le propone al asistente. Por ello, en el dataset de preguntas existen dos tipos de preguntas: i) las respuestas a sintaxis de cron y ii) otras preguntas de aspectos cotidianos donde el modelo deberá descartar la pregunta.

Algunos ejemplos de preguntas y respuestas encontradas en el dataset serán las siguientes.

- ”Pregunta”: ”¿Cómo sería la expresión cron para ejecutar una tarea todos los días a las 9 de la mañana?” ; ”Respuesta”: ”0 9 * * *”
- ”Pregunta”: ”¿Ganó ayer el Real Madrid?” ; ”Respuesta”: ”Irrelevante”

Existen distintos LLM conocidos que se pueden utilizar para la realización de esta tarea. De entre aquellos open source que se barajan para esta tarea, destacan dos: Mistral 7B, desarrollado por la empresa Mistral AI[1] y distintas versiones de Llama 2 13B desarrollado por Meta[2]. El sufijo 7B y 13B se debe al número de parámetros que tiene cada modelo, en este caso 7 y 13 billones respectivamente. El modelo elegido fue el de Mistral debido a su mayor rendimiento (figura 5.5) a pesar de tener menos parámetros en las pruebas de rendimiento o benchmarks, donde Mistral supera claramente con Llama. Además, Mistral supera a Llama tras realizar un fine-tuning a ambos modelos[3] como se muestra en la figura 5.6.

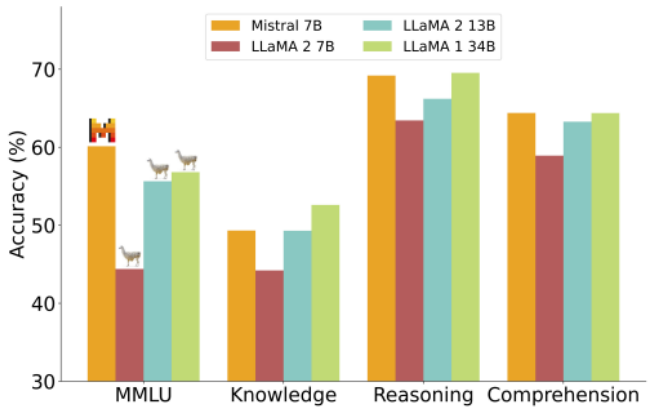

Figura 5.5: Benchmark Llama vs Mistral. Imagen obtenida de [JSM+23]

[1] https://mistral.ai/
[2] https://llama.meta.com/llama2/
[3] A fecha 10/10/2023

Model	Chatbot Arena ELO Rating	MT Bench
WizardLM 13B v1.2	1047	7.2
Mistral 7B Instruct	**1031**	**6.84 +/- 0.07**
Llama 2 13B Chat	1012	6.65
Vicuna 13B	1041	6.57
Llama 2 7B Chat	985	6.27
Vicuna 7B	997	6.17
Alpaca 13B	914	4.53

Figura 5.6: Llama vs Mistral tras hacer fine-tuning. Imagen obtenida de [JSM+23]

Para el entrenamiento de este modelo se ha utilizado la plataforma Google Colab[4], la cual permite ejecutar programas con el lenguaje Python en el navegador sin configuración adicional. Además, ofrece la utilización de GPUs necesarias para el entrenamiento de modelos de redes neuronales de manera más rápida sin coste adicional. La utilización de Google Colab facilita el trabajo en desarrollo de IA ya que permite que el desarrollador utilice su ordenador sin la carga de entrenar un modelo que limitaría el rendimiento del mismo para desarrollar otras tareas en paralelo. Colab utiliza archivos con extensión ipynb, las cuales mezclan bloques de código y de texto con la sintaxis Markdown, que permite organizar el archivo de una mejor manera.

Además de la utilización de Google Colab, se ha utilizado Hugging Face[5] una plataforma especializada en el desarrollo de modelos de machine learning. Hugging Face es open source y además funciona como un repositorio Git especializado en el almacenamiento de modelos de aprendizaje automático. Tendrá dos funcionalidades, por un lado, se accederá al repositorio de código abierto donde se almacena el modelo de Mistral para su descarga y además, se almacenará el nuevo modelo una vez se le haya realizado el fine-tuning para poder acceder de manera pública. En este proceso, cabe destacar que no será necesario descargar el modelo de Mistral, entrenarlo y subir en un nuevo repositorio el modelo de Mistral reentrenado, el cuál utiliza bastante memoria. El enfoque es subir únicamente los nuevos archivos con los datos de los tensores del modelo. Para conocer el modelo base, se añade una referencia al mismo y así se optimiza el almacenamiento en la plataforma.

Por último, como tecnología en el entrenamiento se ha utilizado wandb[6], la cual es una plataforma diseñada para el seguimiento del entrenamiento de modelos de machine learning y la visualización de los resultados de la misma. De esa forma, con los cambios en los hiperparámetros, se podrá comprobar valores que informan de la calidad del modelo como son la precisión o la pérdida del modelo (figura 5.7).

[4] `https://colab.research.google.com/?hl=es`
[5] `https://huggingface.co/docs/hub/index`
[6] `https://wandb.ai/site`

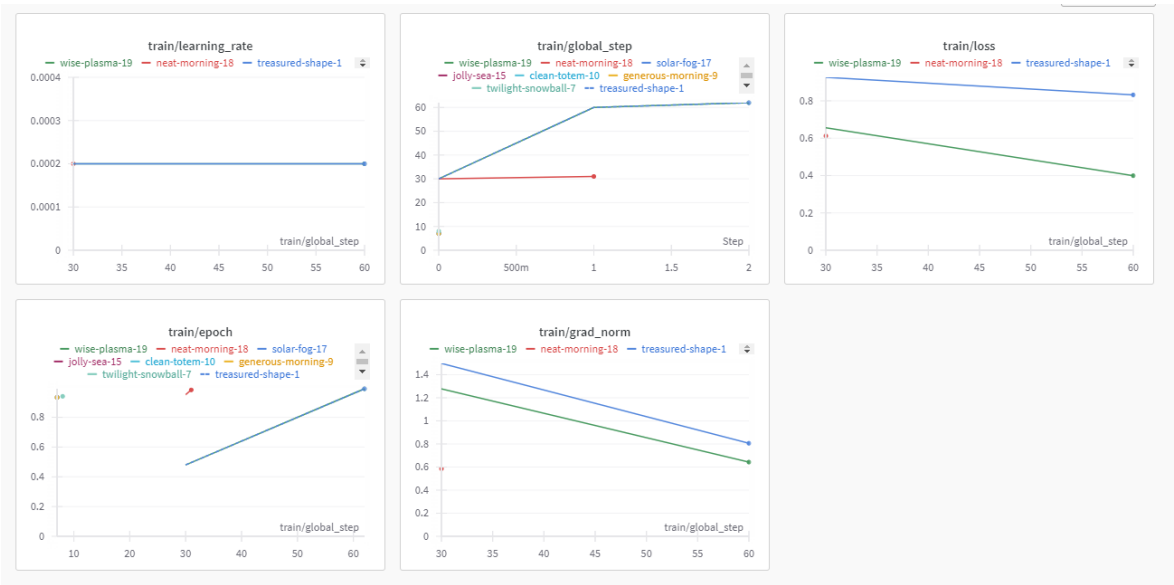

Figura 5.7: Análisis de resultados con wandb

Una vez obtenido el modelo, este se ha almacenado en huggingface para su acceso desde cualquier lugar.

Con el modelo de huggingface desarrollado, se necesita un protocolo para la comunicación con el modelo y que este protocolo sea orgánico con la arquitectura desarrollada. Es por eso que se ha optado por el desarrollo de una API en Python. Se ha tomado la decisión de elegir FastAPI[7] para este propósito. FastAPI es un framework web moderno y sencillo para el desarrollo de APIs con Python. El servidor web resultante tendrá únicamente un endpoint para hacerle una pregunta al LLM, el servido web obtendrá la pregunta, realizada (escrita en el body de la petición), se le entregará al LLM y se devolverá la respuesta de este.

5.2.5 Frontend

La interfaz del usuario ha sido el último microservicio desarrollado y es uno de los más importantes al facilitar la interacción del usuario con la aplicación.

Existen dos enfoques para el desarrollo de un frontend. Por un lado un frontend para cada microservicio descomponiendo de esa forma toda la User Interface (UI) en pequeños módulos autónomos, este enfoque se conoce popularmente como micro-frontend [PMT21]. Sin embargo, este enfoque añade una capa de complejidad en la integración entre los componentes. La arquitectura de micro-frontend puede ser interesante si en la aplicación trabajan distintos grupos, de esa forma, cada grupo podría centrarse en una parte de la aplicación. En este caso, se ha descartado, puesto que no tiene tanto provecho teniendo en cuenta que la aplicación la está desarrollando una única persona.

[7] `https://fastapi.tiangolo.com/`

El segundo enfoque, y el elegido para este proyecto, es el de desarrollar un único frontend integrando todos los componentes para que el usuario sea capaz de programar tareas. Se ha decidido tomar este enfoque pues la aplicación no requiere de muchas pestañas ni de una modularidad tan fina, lo que simplifica significativamente el proceso de desarrollo y mantenimiento. Además, un único frontend permite una experiencia de usuario más coherente y un desarrollo más directo, adaptándose mejor a los recursos disponibles y al alcance del proyecto.

Existen diversos frameworks para el desarrollo de una aplicación, los cuales simplifican el desarrollo de la misma ofreciéndo herramientas adicionales. Entre estos frameworks se destacan React[8] de Meta y Angular[9] o Flutter[10] de Google. Los tres podrían ser perfectamente válidos para el desarrollo de la aplicación; sin embargo, hay una diferencia principal entre ellos: React y Angular utilizan el lenguaje JavaScript, mientras que Flutter utiliza Dart.

Se ha tomado la decisión de elegir Flutter por encima de React o Angular, ya que el framework es conocido por el desarrollador de este proyecto y, además, ofrece ciertas ventajas, como la posibilidad de crear la aplicación con soporte para varias plataformas. Si el cliente lo decidiera así, se podría crear no solo como aplicación web, sino también como aplicación móvil o de escritorio.

El concepto más importante dentro de Flutter son los StatefulWidget, un tipo de widget que puede cambiar su apariencia y comportamiento mientras la aplicación está en uso. Esto es útil para elementos que necesitan actualizarse, como una lista o un contador. Además, para el desarrollo se necesita la biblioteca Flutter Material la cual proporciona un conjunto de widgets predefinidos basados en el diseño de Material Design de Google con los cuales se simplifica la creación de los componentes. Los componentes de Flutter se basan en dos patrones de diseño principalmente: el patrón builder y el patrón MVC (sección 5.6)

La aplicación se compone de seis StatefulWidgets, cada una de ellas siendo una página de la aplicación. Primero, te encuentras con la pestaña de Login, en la cual el usuario debe introducir sus credenciales para acceder a la aplicación. Al ser una aplicación cerrada, se deben dar las credenciales a los clientes que contraten este servicio, por tanto, no existe una manera de registrarte, para suplir esta carencia, se ha especificado cuál es el procedimiento a seguir para obtener las credenciales. Una vez se comprueban las credenciales del usuario, si estas son erróneas se despliega un mensaje de error y si son correctas accedes a la aplicación, los datos del login se almacenan en tiempo de ejecución mediante el patrón Singleton (sección 5.6).

[8] `https://es.react.dev/`
[9] `https://angular.io/`
[10] `https://flutter.dev/`

Al entrar en la aplicación, la primera sección a la que se accede es la referente a las tareas. Esta sección se conforma con tres páginas, inicialmente, encontraras una lista con las tareas definidas por el usuario (figura 5.8) y con unos pocos datos de interés como son el nombre, la fecha de creación y el estado. Además, se ofrece la posibilidad de crear una tarea, lo cual llevaría a la segunda página. Si se pulsa en la tarea, te llevaría a la tercera página donde se ve en mayor detalle los atributos de la tarea.

Figura 5.8: Pestaña Tareas

En esta segunda página, se debe rellenar un formulario indicando los campos necesarios para la creación de la tarea desde el nombre y descripción, pasando por los datos de la llamada HTTP hasta la definición de la programación de la tarea.

En la tercera página se listan los datos de la tarea (figura 5.9), pudiendo modificarse estos datos o también se podría borrar la tarea de la base de datos. En el final de la página existe un botón con el que se muestran los eventos generados por la tarea en la base de datos de Cockroach DB.

En las dos páginas anteriores, en la sección de la programación de la tarea, se encuentra un botón de ayuda para acceder al asistente virtual con el que mediante lenguaje natural se podrá obtener una expresión cron según las necesidades indicadas.

La segunda sección consta de los datos de los usuarios; en el menú lateral, si se accede a la sección de configuración se pueden encontrar los datos personales del usuario y además, se puede cambiar la contraseña de usuario. También se puede eliminar la cuenta, borrando de esa forma los datos del usuario y las tareas asociadas.

Además, si el usuario es de tipo administrador, se le mostrará como título en la barra de aplicación y además se le habilitará la opción del panel de administración en la barra lateral. En este panel de administración podrá crear nuevos usuarios y además modificar los datos de los usuarios ya existentes, incluyendo cambios de contraseña y cambio en los privilegios de los usuarios.

Figura 5.9: Pestaña Información de la tarea

En los siguientes diagramas de secuencia (figuras 5.10 y 5.11) se pueden encontrar los casos de uso de la aplicación. Cabe aclarar que por simplificación se agrupa a todo el backend en una entidad aunque existen varios servicios que integran dicho backend y además las funciones que puede realizar un usuario las puede hacer un administrador pero no ocurre de la misma forma con las funciones de un administrador que no puede realizarlas un usuario.

Por último, para el despliegue de la aplicación, en Flutter existe el comando ”flutter build web” con el cuál se compila la aplicación de Flutter en una aplicación web, comprimiendo de esa forma el tamaño de la aplicación al utilizar archivos Javascript, CSS y HTML. Una vez se han generados los archivos, se deben desplegar mediante un servidor web. Un servidor web es un sistema que entrega páginas a los usuarios mediante los protocolos HTTP y HTTP. Se ha elegido nginx como servidor web, el cual es conocido por su eficiencia y su escalabilidad para este propósito. En la sección 5.5 se dará una definición más detallada de nginx.

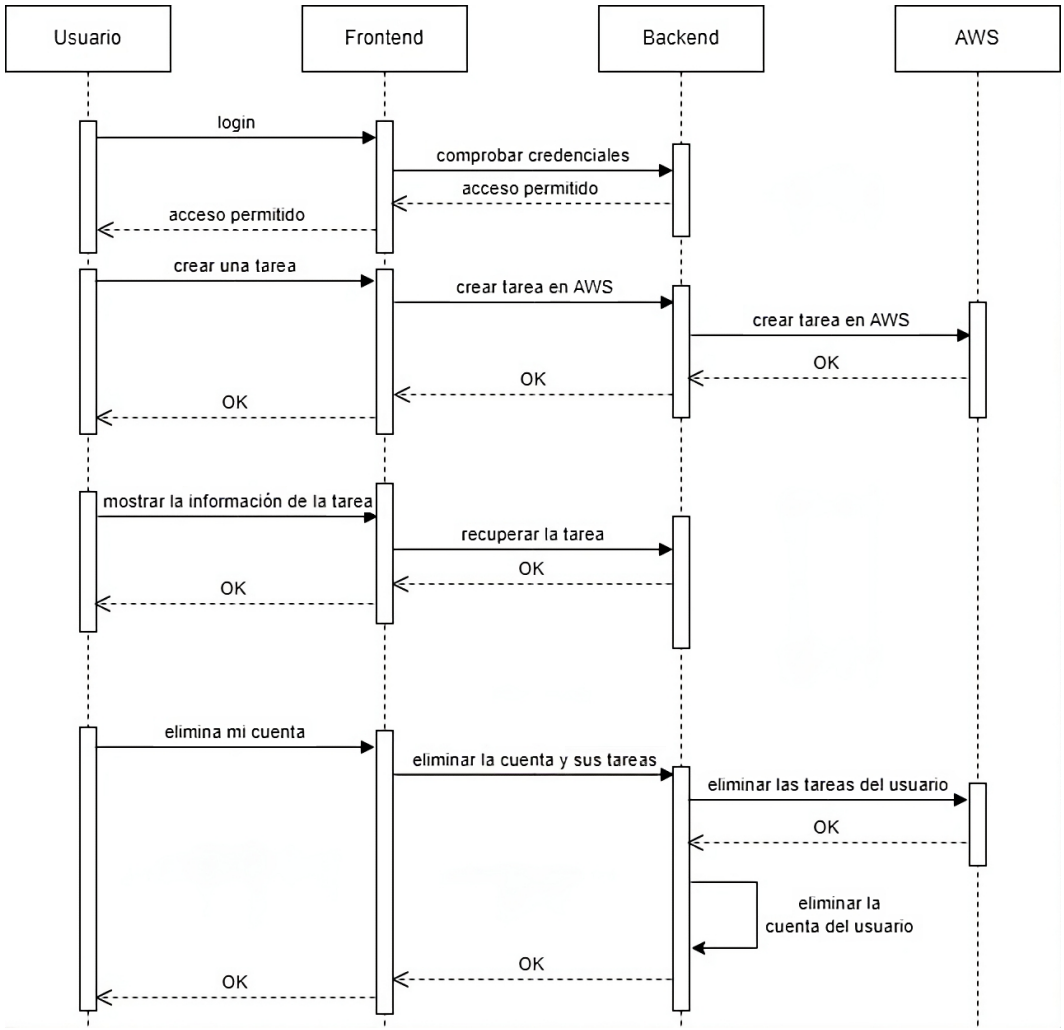

Figura 5.10: Diagrama de secuencia usuario

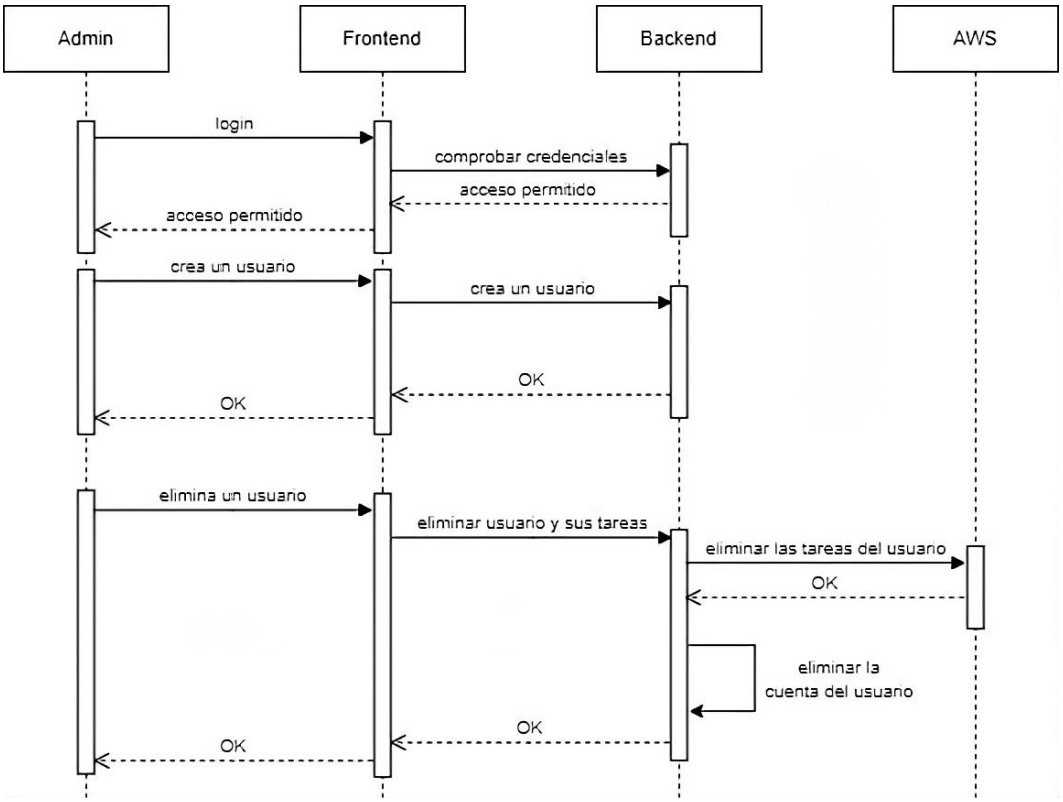

Figura 5.11: Diagrama de secuencia administrador

5.3 Servicios AWS

AWS ha sido el servicio de computación en la nube elegido para la programación de las tareas. Como se explicó la sección 3.2.2, AWS-EB es el servicio elegido para la programación de tareas. Esta decisión se basa en la idoneidad de AWS-EB para las necesidades del proyecto en cuestión, en comparación con alternativas como Google Cloud o Microsoft Azure. La elección de AWS se fundamenta en varios factores. En primer lugar, AWS es una de las empresas pioneras en el campo de la computación en la nube y ha consolidado su posición como uno de los líderes de la industria. AWS ofrece una amplia gama de recursos, incluyendo extensas guías de desarrollo y una activa comunidad de usuarios que proporciona soporte a través de foros en línea, lo que facilita la resolución de problemas y la optimización del desarrollo. Por último, los resultados de un estudio comparativo han demostrado que el costo de AWS-EB es más competitivo en comparación con otras alternativas, al tiempo que ofrece un rendimiento igual o superior. En comparación a Google Cloud, es más sencillo de utilizar pues solamente se necesita un servicio para la programación de estas tareas mientras que en Google se necesita GCTK y GCSCH. Por otro lado, Microsoft Azure si bien ofrece un servicio similar, su soporte es menos competitivo que el de AWS.

Además, al ser una de las empresas más importantes, AWS cuenta con numerosas guías de desarrollo y una amplia comunidad con la cual se puede interactuar a través de foros para resolver incidencias. Por último, los resultados del estudio muestran que el precio de AWS-EB es más económico que otras alternativas ofreciendo el mismo o mayor rendimiento.

Las tareas de AWS-EB se configuran para disparar una acción cuando se cumpla el horario especificado. En este caso, se dispara una función de lambda de AWS. AWS Lambda[11] es un servicio sin servidor, elimina la necesidad de preocuparse por la infraestructura, permitiéndote enfocarte completamente en el código del programa. Esta basado en eventos, por lo que permite ejecutar código en respuesta a una amplia gama de eventos, desde cambios en los datos hasta solicitudes HTTP. Esto significa que se puede construir y desplegar aplicaciones y servicios backend que se ejecutan en servidores que se activan únicamente para ejecutar la función y, a continuación, se desactivan cobrando únicamente por el tiempo de ejecución de la función.

Las tareas deben estar asociadas de un rol especificando los permisos que puede realizar la misma. De esa forma, se puede dar permiso limitando las acciones que puede realizar una tarea creada en AWS-EB en el caso de que se descubra el valor de este rol. En las tareas creadas se les ha dado el rol de "AWSLambda_FullAccess" el cual permite control absoluto de funciones Lambda. Se podría pensar en darle otros roles, como el de "AWSLambdaExecute"; sin embargo, AWS no es demasiado específico con los nombres ya que en su descripción esta Lambda solo serviría para acceder a los recursos de AWS S3, un servicio de almacenamiento.

[11] `https://aws.amazon.com/es/lambda/`

La función que ejecutará Lambda será una escritura en una base de datos externa. Este nuevo registro en la base de datos contará con una serie de parámetros que se definen al crear la tarea. El objetivo es demostrar la funcionalidad de la aplicación. Esta demostración sirve como punto de partida para la futura modificación para llamar a un endpoint externo con una tarea en segundo plano. Los registros que se almacenan son una Uniform Resource Locators (URL), el método HTTP y el cuerpo de la petición, que son los parámetros estándar al hacer una llamada HTTP.

El lenguaje de programación utilizado para la Lambda ha sido Python[12], su facilidad de uso y la disponibilidad de bibliotecas que permiten la comunicación con bases de datos ha sido clave para elegir esta tecnología.

Existen diversas alternativas para la elección de la base de datos para resolver esta funcionalidad. Por un lado, existen plataformas como CockroachDB[13] y MongoDB[14] con las que se puede desplegar una base de datos. Se busca una plataforma que permita el despliegue de la base de datos de manera gratuita y que sea accesible a través de Internet desde cualquier punto. Ambas cumplen con esta definición. Sin embargo, se ha elegido CockroachDB pues es una base de datos de tipo relacional y se quiere mantener la misma arquitectura de base de datos con el resto de microservicios desarrollados. En esta base de datos se almacenarán los registros y permitirá observar la correcta ejecución de las tareas creadas en AWS-EB. En esta base de datos de Postgres existirá únicamente la tabla ”events” 5.3 en la que se almacena la información mencionada referente a la llamada HTTP y además algunos campos relevantes para poder extraer la información y tener un mayor control, estos campos son el id del usuario, el id de la tarea en AWS-EB y la fecha en la que se ha escrito el registro.

5.4 Kubernetes

5.4.1 Conceptos generales

K8s[15] es una plataforma open source de orquestación de contenedores desarrollada por Google. Desde su creación en 2014, K8s ha crecido hasta convertirse en un estándar en el despliegue de aplicaciones en la nube [HBB17]. K8s otorga una serie de ventajas que elevan una aplicación a ser más escalable y tolerante a fallos. En el desarrollo con Kubernetes, existen diversos elementos que abstraen al programador de la lógica para centrarse en el desarrollo y la definición de los requisitos de hardware que se necesitan. La sintaxis que utiliza K8s es declarativa en la que se define el estado que se desea para el sistema y por debajo K8s gestiona las operaciones para llegar a ese punto.

[12] https://www.python.org/
[13] https://www.cockroachlabs.com/
[14] https://www.mongodb.com/
[15] https://kubernetes.io/docs/concepts/overview/

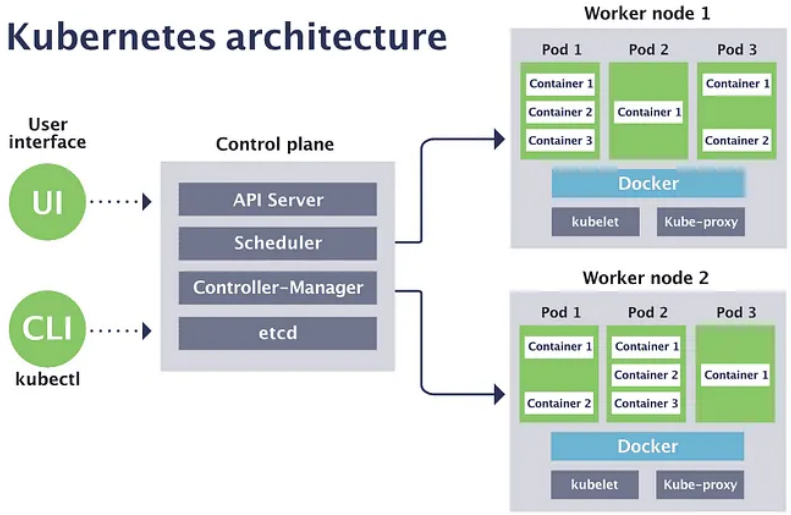

Figura 5.12: Arquitectura de Kubernetes. Imagen obtenida de Medium

En la figura 5.12 se encuentra la arquitectura general de Kubernetes; se le suele denominar cluster a la colección de nodos para ejecutar una aplicación. Existen dos tipos de nodos, los master y los worker. Se considera que K8s es escalable ya que permite el añadir tantos nodos como se necesite. Estos nodos representan diferentes máquinas tanto físicas como virtuales, añadiendo más potencia de computación al sistema. En la figura 5.12 el nodo master es el que se le define como Control plane, es el nodo que se encarga de administrar el estado del cluster y tiene cuatro componentes.

- *Api server.* Es el punto central de comunicación entre los nodos.
- *Etcd.* Es una base de datos con el estado del cluster.
- *Controller manager.* Se encarga de ejecutar procesos que regulan el estado del sistema
- *Scheduler.* Es un programador de tareas en segundo plano (sección 3.1).

Por otro lado, los nodos trabajadores que son los encargados de la ejecución de los contenedores. Dentro de un nodo trabajador existen diversos elementos que se necesitan para poder desplegar un contenedor. En primer lugar, se necesita un Pod, este pod, es una instancia de la aplicación *contenerizada*. Por encima del Pod, existe una capa denominada Deployment, la cual te permite desplegar un conjunto de Pods.

Cuando se quiere exponer un servicio con Kubernetes, se debe declarar un elemento Service, que expone la aplicación y permite a otros Pods dentro del cluster comunicarse unos con otros. Existen diversos niveles de exposición, destacan ClusterIP, exponiendo la aplicación dentro del cluster, NodePort, la cual asigna a cada worker node que despliegue la aplicación una IP fija y por último, LoadBalancer, la cual necesita de un servicio externo que gestione un balanceador de carga mejorando el rendimiento de la aplicación.

5.4.2 Arquitectura

En primer lugar, se deben crear las bases de datos y para ello se componen de varias capas de manera que sean bases de datos persistentes incluso si el cluster de Kubernetes y se elimina que la base de datos siga existiendo.

Para una base de datos se debe crear un Storage Class (SC), la cual permite dotar de políticas individuales a cada clase de almacenamiento. A continuación ya se puede crear el Persistent Volume (PV), que representa un volumen de almacenamiento persistente en el clúster y que se le asocia al SC. Después se conecta el PV con el Persistent Volume Claim (PVC), el cual solicita un volumen de almacenamiento persistente y permite el acceso desde un Pod. Por último, ya se puede crear un pod con una imagen de un sistema de gestión de bases de datos conectado al PVC.

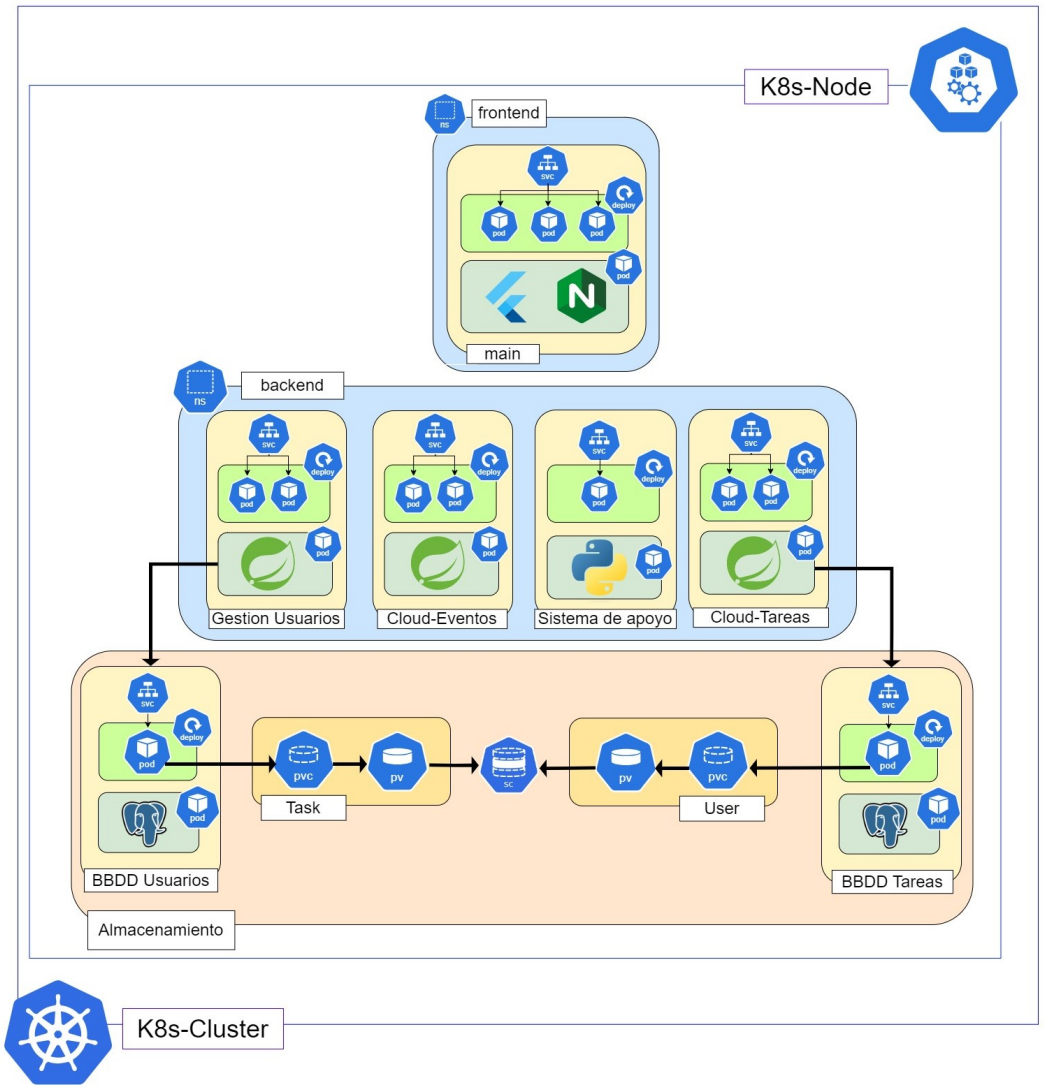

Figura 5.13: Arquitectura en K8s

Con la base de datos en funcionamiento se procede a desplegar las instancias de los microservicios en los diferentes Pods con Deployments y exponerlos mediante Services. En el diagrama 5.13 se describe visualmente la arquitectura dentro de K8s. Es bastante parecido a una arquitectura clásica cliente-servidor, sin embargo, en este caso las aplicaciones son independientes. Existen Deployments con una cantidad de Pods distinta, se ha determinado su número tras realizar una serie de pruebas en las que se ha obtenido un equilibrio entre la capacidad de los recursos hardware y la carga que puede soportar la aplicación.

Al estar trabajando de manera local, aunque el cluster es exportable a un servicio de computación en la nube con mayor potencia, se ha determinado la utilización de un único nodo que actúa como nodo master y worker a la vez.

5.5 Despliegue de la aplicación

A la hora de desplegar la aplicación para exponerla en Internet se deben exponer todos los microservicios, tanto el frontend como el backend.

5.5.1 Despliegue de los contenedores

En el desarrollo de los microservicios se pueden identificar tres fases. Primero el diseño y desarrollo de las aplicaciones, cada una con sus propios frameworks y lenguajes de programación. En segundo lugar, el empaquetamiento de la aplicación en un contenedor. Por último, el despliegue del contenedor en un repositorio público tanto para tener la trazabilidad de las versiones como para poder acceder desde diferentes dispositivos.

El diseño y desarrollo de los diferentes microservicios se ha expuesto detalladamente en la sección 5.2.

En cuanto al empaquetamiento de los microservicios, se ha utilizado Docker[16]. A diferencia de otras alternativas como Podman[17] Docker ofrece una amplia adopción y soporte en la comunidad, lo que facilita el acceso a recursos, documentación y soluciones a problemas comunes. Por otro lado, Docker ofrece una experiencia de usuario más amigable y una interfaz de línea de comandos intuitiva que facilita el empaquetamiento de los microservicios. Para empaquetar los microservicios es necesario elegir una imagen adecuada. Las imágenes en Docker son paquetes ligeros que contienen todo lo necesario para ejecutar una aplicación. En cuanto a los microservicios con Spring Boot, se ha utilizado *openjdk:17-jdk-buster* coincidiendo con el desarrollo en Java-17. Para el microservicio del sistema de apoyo se ha necesitado de *pytorch/pytorch:2.3.0-cuda12.1-cudnn8-runtime*, una imagen bastante pesada, pero que contiene todas las librerías para la utilización de un LLM que necesita de *pytorch* y la aceleración con GPU. Por último, para el frontend, se ha utilizado *nginx:alpine* como imagen para desplegar el servidor web.

[16] `https://www.docker.com/`
[17] `https://podman.io/`

Por último, para la creación y publicación de los contenedores, se ha creado un pipeline con Github Actions[18]. Este pipeline compila los proyectos y empaqueta las aplicaciones en contenedores. Para los microservicios con Spring Boot se compila con la herramienta maven generando el archivo Java ARchive (JAR). Para el frontend, se ejecuta el comando flutter build web obteniendo los archivos estáticos. Después crea el contenedor de la aplicación con Docker, finalmente, se suben estas imágenes a un repositorio en Github Packages[19]. Cada aplicación tiene un repositorio individual, lo que permite mantener la independencia entre los microservicios. Esta separación de repositorios facilita la gestión del ciclo de vida de cada microservicio, permitiendo que se desarrollen y desplieguen de manera autónoma sin interferir con otros componentes del sistema.

Para el manejo de los contenedores, cada uno tiene una etiqueta que representa la fecha de su creación. Esta práctica sirve como un mecanismo de versionado, permitiendo identificar rápidamente la generación de cada contenedor y facilitando la trazabilidad de cambios y actualizaciones.

Además, se utiliza la etiqueta "latest"para indicar cuál es el contenedor más reciente creado. Esta etiqueta proporciona una referencia directa a la versión más actualizada del contenedor, simplificando el proceso de despliegue al asegurar que siempre se esté utilizando la versión más reciente y probada de la aplicación.

Esta estrategia de etiquetado y versionado no solo mejora la claridad y organización en el desarrollo y despliegue de microservicios, sino que también ayuda a prevenir conflictos de versiones y garantiza que los entornos de producción utilicen siempre las versiones más recientes y seguras de los contenedores.

Finalmente, se puede acceder a estos contenedores mediante Kubernetes para así desplegar la aplicación.

5.5.2 Reverse tunnel

Existen diversas alternativas para exponer una aplicación de las características descritas en este proyecto a Internet.

Por un lado, se puede utilizar un servicio de Hosting en la web como puede ser Github Pages[20]. Sin embargo, al necesitar comunicación entre frontend y backend este enfoque no es el adecuado, ya que en este tipo de plataformas solo se pueden ofrecer archivos web estáticos, pero no se podría exponer el backend. La segunda opción, es utilizar un servicio web de Domain Name System (DNS) con algun servicio de computación en la nube, como AWS Route 53[21] o Google Cloud DNS[22].

[18] `https://github.com/features/actions`
[19] `https://github.com/features/packages`
[20] `https://pages.github.com/`
[21] `https://aws.amazon.com/es/route53/`
[22] `https://cloud.google.com/dns?hl=es`

En este caso, sería necesario recurrir a servicios de computación como un Virtual Private Server (VPS) o utilizar un servicio de Kubernetes en la nube, como AWS Elastic Kubernetes Service[23].

Si se opta por desplegar la aplicación en un VPS, se pueden aprovechar las instancias gratuitas que ofrece Amazon por tiempo limitado. Sin embargo, estas instancias no proporcionan la potencia suficiente para soportar los recursos que requiere Kubernetes. Se estima que se necesitarían al menos cuatro gigabytes de RAM, mientras que el máximo ofrecido es de dos gigabytes. En el caso de usar un servicio de Kubernetes el precio se dispararía ya que este tipo de servicios están en ejecución continuamente con un precio de 0.10 $ la hora[24]. En ambos casos, habría que sumar el precio del registro del dominio, anualmente sobre los diez euros.

La tercera opción, es utilizar un reverse tunnel, un mecanismo de red que permite establecer una conexión desde una máquina en una red privada a otra máquina en una red pública. Al contrario del modo tradicional donde la comunicación se realiza desde la máquina privada a la máquina pública, en este caso, la comunicación es desde la pública a la privada. Es decir, se puede exponer un puerto del localhost de un ordenador asignandole un nombre de dominio y exponiendolo a Internet para que cualquier persona pueda acceder.

En este campo, aunque existen muchas alternativas; ngrok[25] y loophole[26] destacan en este servicio. Ambas plataformas permiten la exposición a Internet de una aplicación mediante el protocolo HTTPS, consiguiendo una conexión segura y en ambas existe la misma limitación, en el plan gratuito solo se puede exponer un único puerto. Se ha decidido utilizar ngrok al ser una plataforma con una mayor madurez que ofrece nombres de dominio estáticos para la aplicación. En este caso, el nombre de dominio, si bien es estático, se genera automáticamente por la plataforma y en el caso de esta aplicación, se ha obtenido el nombre de dominio ”adapted-first-snipe.ngrok-free.app”.

Al exponerse la aplicación mediante el protocolo HTTPS, ha sido necesario elevar la seguridad del resto de microservicios, ya que ciertos navegadores no permiten la comunicación entre los protocolos HTTP y HTTPS. Por ello, se han creado certificados autogenerados mediante la herramienta keytool. De esa forma, se asegura que la conexión está cifrada y no se pueden acceder a los datos por ningún tipo de software malicioso.

5.5.3 Reverse proxy

Una vez se tiene el método para exponer la aplicación a Internet hay que pensar en como resolver la limitación de desplegar todos los microservicios teniendo en cuenta que solo se puede exponer un puerto a través de ngrok.

[23] `https://aws.amazon.com/es/eks/`
[24] `https://aws.amazon.com/es/eks/pricing/`
[25] `https://ngrok.com/`
[26] `https://loophole.cloud/`

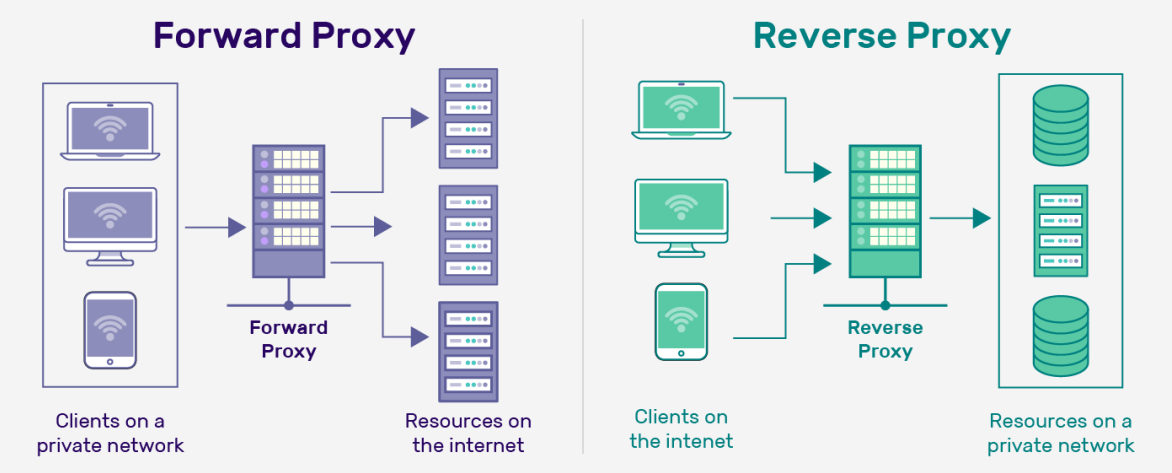

Figura 5.14: Reverse proxy vs Forward proxy. Imagen obtenida de Security Boulevard

Para resolver esta limitación, surgen los ”reverse proxy”, un reverse proxy es un servidor que actúa como intermediario en la comunicación del cliente con una serie de servidores de contenido alojados en una red privada, los contenidos vienen proporcionados por servidores web, tanto los servicios de backend como los de frontend. Para determinar a que servidor debe enviar la petición, se puede configurar un reverse proxy para acceder a diferentes contenidos mediante el path de una URL, las cookies o los encabezados de la petición HTTP. Un cliente envía la solicitud a una URL, pensando que es un servidor, aunque en realidad sea el reverse proxy, dependiendo de los datos mencionados, se reenvía la solicitud al backend, se obtiene la solicitud y se devuelve al cliente.

Este reverse proxy es contrario al tradicional forward proxy que utilizan los ordenadores personales al conectarse a Internet. En los forward proxy, el servidor actua como intermediario entre el cliente y los servidores de contenido públicos en Internet, controlando así los contenidos a los que se accede. En la interacción, el cliente envía la solicitud al forward proxy y después este a los servidores de contenido externos (figura 5.14).

En este campos surgen diferentes tecnologías para el desarrollo de un reverse proxy. Nginx[27] o traefik[28] son opciones para el desarrollo de esta sección. La diferencia principal es el enfoque: nginx es una tecnología más robusta y configurable mientras que traefik es una gran opción para entornos de microservicios, permitiendo su integración con Docker y Kubernetes. Por la naturaleza de esta tarea parece que traefik puede ser la opción ideal. Sin embargo, se quiere desacoplar cada componente de la aplicación para permitir posteriores cambios de la manera más sencilla posible. Por este motivo, ya que traefik dependería de Kubernetes, se opta por utilizar nginx.

[27] https://nginx.org/en/
[28] https://traefik.io/

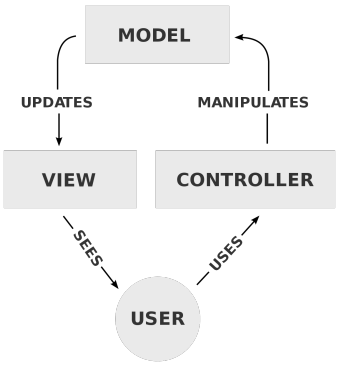

Figura 5.15: Patrón MVC. Imagen obtenida de Medium

5.6 Patrones de diseño

En esta sección se definen los patrones de diseño utilizados en el proyecto. Los patrones de diseño son soluciones probadas y documentadas para problemas comunes que surgen en el diseño de software [Uza22]. Estas soluciones encapsulan las mejores prácticas y experiencias acumuladas por desarrolladores a lo largo del tiempo, promoviendo la reutilización de código, la modularidad y la escalabilidad.

5.6.1 Model–view–controller (MVC)

El patrón MVC es un patrón de diseño ampliamente utilizado en el desarrollo de interfaces [Buc09]. Se divide en tres componentes principalmente (figura 5.15).

- Modelo. El modelo representa los datos y la lógica de negocio de la aplicación. Es responsable de almacenar y manipular los datos de la aplicación, así como de realizar operaciones relacionadas con la lógica de negocio.
- Vista. La vista es la capa de presentación de la aplicación. Es responsable de mostrar los datos al usuario y de presentar la interfaz de usuario con la que el usuario interactúa.
- Controlador. El controlador actúa como un intermediario entre el modelo y la vista. Se encarga de manejar las interacciones del usuario y de actualizar tanto el modelo como la vista en consecuencia.

En Flutter, las clases de datos, por ejemplo Usuario, definen el modelo; los widgets se encargan de conformar la vista para la presentación con el usuario y los StatefulWidget como controlador para que el usuario interaccione con la aplicación.

5.6.2 Builder

El patrón builder es un patrón de diseño creacional que se utiliza para construir objetos complejos paso a paso [Fre15]. Proporciona una forma flexible de construir diferentes tipos de objetos utilizando el mismo proceso de construcción (figura 5.16).

La idea básica detrás del patrón Builder es separar la construcción de un objeto complejo de su representación, de modo que el mismo proceso de construcción pueda crear diferentes representaciones del objeto.

El patrón Builder consta de varios componentes clave que trabajan juntos para construir objetos complejos de manera flexible. En primer lugar, la clase Builder establece una interfaz común para la creación de partes del objeto. Luego, la clase Concreta Builder implementa esta interfaz y proporciona métodos concretos para configurar y ensamblar las partes del objeto. Por último, el objeto Producto representa el resultado final de la construcción, pudiendo ser una entidad simple o compleja con múltiples partes.

Este patrón se ha utilizado con Flutter para construir diferentes objetos dependiendo de las necesidades, por ejemplo, ciertos campos son visibles o tienen distinto estilo durante el registro de una tarea pero no durante la modificación de su información.

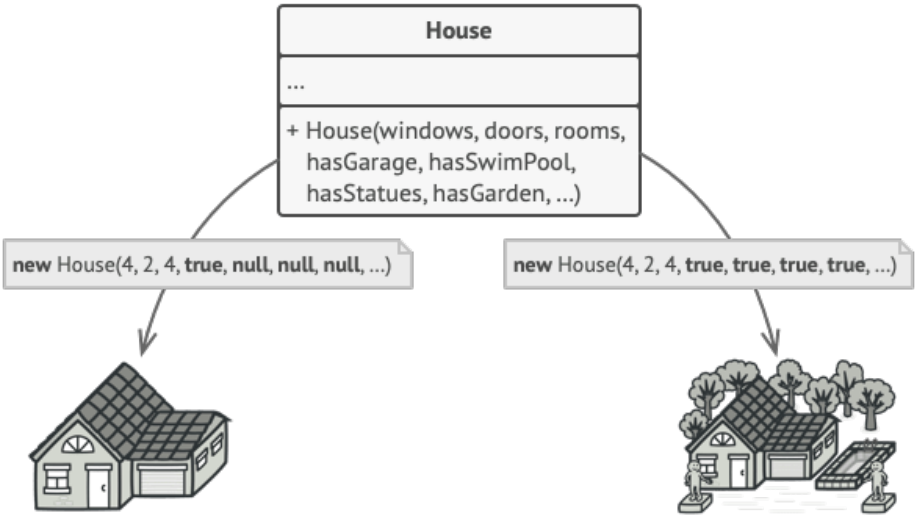

Figura 5.16: Patrón Builder. Imagen obtenida de Refactoring.Guru

5.6.3 Singleton

El patrón Singleton es un patrón de diseño creacional que garantiza que una clase tenga una única instancia y proporciona un punto de acceso global a esa instancia [Mus23]. Esto significa que, una vez que se crea la primera instancia de la clase, todas las solicitudes de instancias posteriores devolverán la misma instancia.

Este patrón es útil cuando se desea que una clase tenga una única instancia en toda la aplicación (figura 5.17), como por ejemplo, para manejar la configuración global, la conexión a la base de datos o el registro de eventos. Para implementar el patrón Singleton, se utiliza un constructor privado para evitar que se creen instancias adicionales de la clase desde fuera de la clase misma. Luego, se proporciona un método estático para acceder a la única instancia de la clase, creándola si aún no existe.

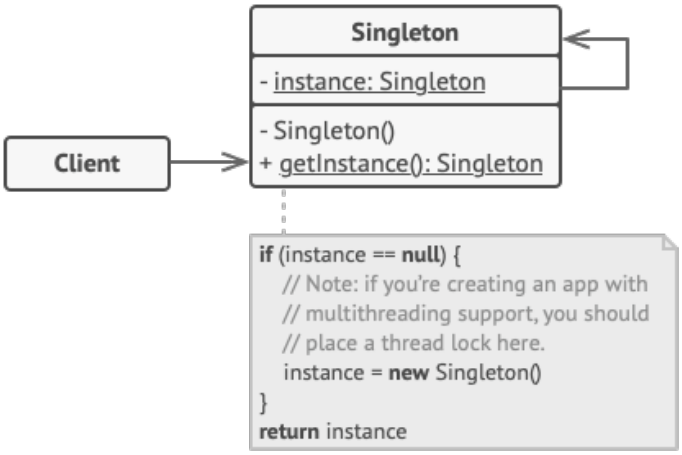

Figura 5.17: Patrón Singleton. Imagen obtenida de Refactoring.Guru

Sin embargo, el patrón Singleton también se considera un anti-patrón en muchos contextos debido a varios inconvenientes.

- *Dificultad para realizar pruebas unitarias.* Este patrón puede complicar ciertos tipos de pruebas unitarias ya que introducen un estado global en la aplicación, lo que puede llevar a dependencias implícitas entre las pruebas y dificultar el aislamiento de las mismas.
- *Mantenimiento.* El uso de Singletons puede llevar a un diseño rígido y difícil de modificar, ya que el patrón establece una dependencia global que puede ser difícil de refactorizar o reemplazar más adelante.
- *Problemas de concurrencia.* En entornos concurrentes, la implementación de Singletons debe ser cuidadosamente diseñada para manejar el acceso simultáneo de múltiples hilos, lo que puede complicar el código y reducir su eficiencia.

A pesar de estos inconvenientes, el patrón Singleton ha sido empleado en el diseño del frontend para almacenar datos temporales sin que puedan ser modificados por distintas clases al mismo tiempo, creando un desajuste. Esta aplicación específica del patrón puede ser ventajosa para gestionar estados compartidos de manera controlada.

5.6.4 Adapter

El patrón Adapter es un patrón de diseño estructural que permite que dos interfaces incompatibles trabajen juntas al actuar como un intermediario entre ellas [Sar16]. Consiste en crear una clase adaptadora que convierte la interfaz de una clase en otra interfaz que un cliente espera. Esta adaptación permite que las clases con interfaces incompatibles cooperen sin cambiar su código fuente.

En los microservicios con Java, Java Persistence API (JPA) implementa el patrón Adapter al proporcionar una interfaz común para interactuar con diferentes proveedores de persistencia de datos, en este caso con Postgres. JPA actúa como un adaptador entre la interfaz orientada a objetos de la aplicación y la base de datos relacional, abstrayendo las complejidades del acceso a la base de datos y el mapeo entre objetos y relaciones.

5.6.5 Dependency injection

El patrón de Inyección de Dependencias es un patrón de diseño en el que los componentes de una aplicación no crean las dependencias que necesitan para funcionar, sino que se las suministran desde fuera [SvD19]. Esto significa que las dependencias de un componente se *inyectan* en él desde el exterior, en lugar de que el propio componente las cree internamente.

En los microservicios con Spring Boot, la Inyección de Dependencias se implementa mediante el contenedor de Spring Inversion of Control (IoC). Spring Boot utiliza anotaciones como @Autowired para inyectar automáticamente las dependencias en los componentes de la aplicación.

El microservicio de apoyo también implementa este patrón, FastAPI, implementa el patrón de Inyección de Dependencias a través de la biblioteca Dependecy Injection de Pydantic. FastAPI utiliza este enfoque para permitir la inyección de dependencias en los controladores de la API.

5.6.6 Controller

El patrón Controller es un patrón de diseño arquitectónico que se utiliza comúnmente en el desarrollo de aplicaciones web para separar la lógica de negocio de la lógica de presentación [Sar16]. En este patrón, los controladores son responsables de recibir las solicitudes del cliente, procesarlas y devolver una respuesta adecuada. Los controladores actúan como intermediarios entre el cliente y el resto de la aplicación, manejando la entrada del usuario y coordinando las acciones correspondientes.

En Spring Boot, los controladores se implementan utilizando anotaciones como @RestController y @Controller. Estas anotaciones permiten que una clase sea identificada como un controlador y define los endpoints de la API web.

FastAPI también implementa los controladores. En este caso se utilizan las anotaciones @app.get o @app.post para indicar los endpoints de la API web.

Capítulo 6

Resultados

En esta sección se detallarán los resultados obtenidos en el desarrollo de la aplicación para la planificación de tareas en entornos cloud. También se exponen diversas estadísticas para mostrar la magnitud y complejidad del proyecto creado. Por último, se realizará un estudio del coste y tiempo empleado en el desarrollo del proyecto.

El código fuente de la aplicación se puede encontrar en la siguiente organización de Github. Al ser necesarios distintos repositorios para mantener la independencia entre los distintos microservicios se ha optado por crear la organización que cuenta con cinco repositorios de código para cada microservicio. Además, existen cinco repositorios de paquetes de Github, donde se almacenan las imágenes de los microservicios. Por último, se ha creado un repositorio con los archivos de configuración en K8s para el despliegue de la aplicación

- `https://github.com/Forte-GestorJobs`
- `https://github.com/Forte-GestorJobs/despliegue`

En el cuadro 6.1 se encuentran los repositorios de código mientras que en el cuadro 6.2 se encuentran los repositorios de paquetes.

Cuadro 6.1: Repositorios de código

Microservicio	**Enlace**
Frontend	`https://github.com/Forte-GestorJobs/micro-frontend-main`
Gestión de Usuarios	`https://github.com/Forte-GestorJobs/micro-gestion-usuarios`
Gestión de Tareas	`https://github.com/Forte-GestorJobs/micro-cloud-tareas`
Gestión de Eventos	`https://github.com/Forte-GestorJobs/micro-cloud-eventos`
Sistema de Apoyo	`https://github.com/Forte-GestorJobs/micro-sistema-apoyo`

Cuadro 6.2: Repositorios de paquetes

Microservicio	Enlace
Frontend	https://github.com/Forte-GestorJobs/micro-frontend-main/pkgs/container/micro-frontend-main
Gestión de Usuarios	https://github.com/Forte-GestorJobs/micro-gestion-usuarios/pkgs/container/micro-gestion-usuarios
Gestión de Tareas	https://github.com/Forte-GestorJobs/micro-cloud-tareas/pkgs/container/micro-cloud-tareas
Gestión de Eventos	https://github.com/Forte-GestorJobs/micro-cloud-eventos/pkgs/container/micro-cloud-eventos
Sistema de Apoyo	https://github.com/Forte-GestorJobs/micro-sistema-apoyo/pkgs/container/micro-sistema-apoyo

6.1 Resultado Final

Para acceder a la aplicación, el primer paso es iniciar sesión en la aplicación con las credenciales que hayan sido concedidas por el administrador del sistema (figura 6.1). En caso de que no se tengan las credenciales o se hayan olvidado el nombre de usuario o la contraseña, se deberán solicitar mediante el correo suministrado.

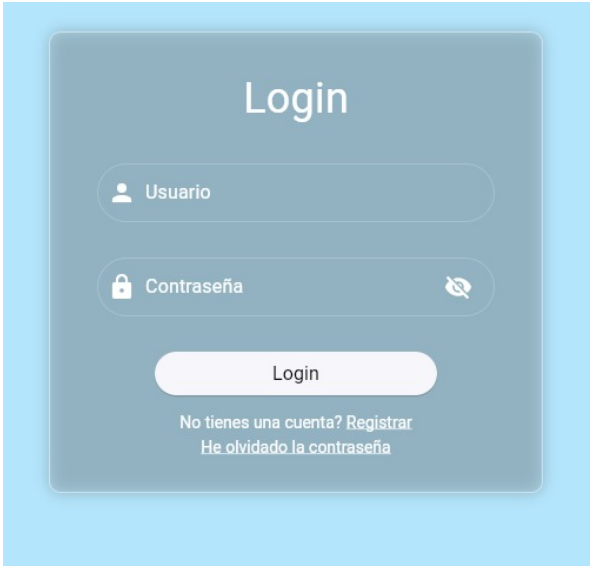

Figura 6.1: Pestaña inicio de sesión

Una vez accedes a la aplicación, se encuentra la pestaña Tareas donde se pueden listar, eliminar y crear nuevas tareas (figura 6.2). Las tareas aparecen con una serie de datos relevantes como son el nombre, la fecha de creación y el estado de la tarea.

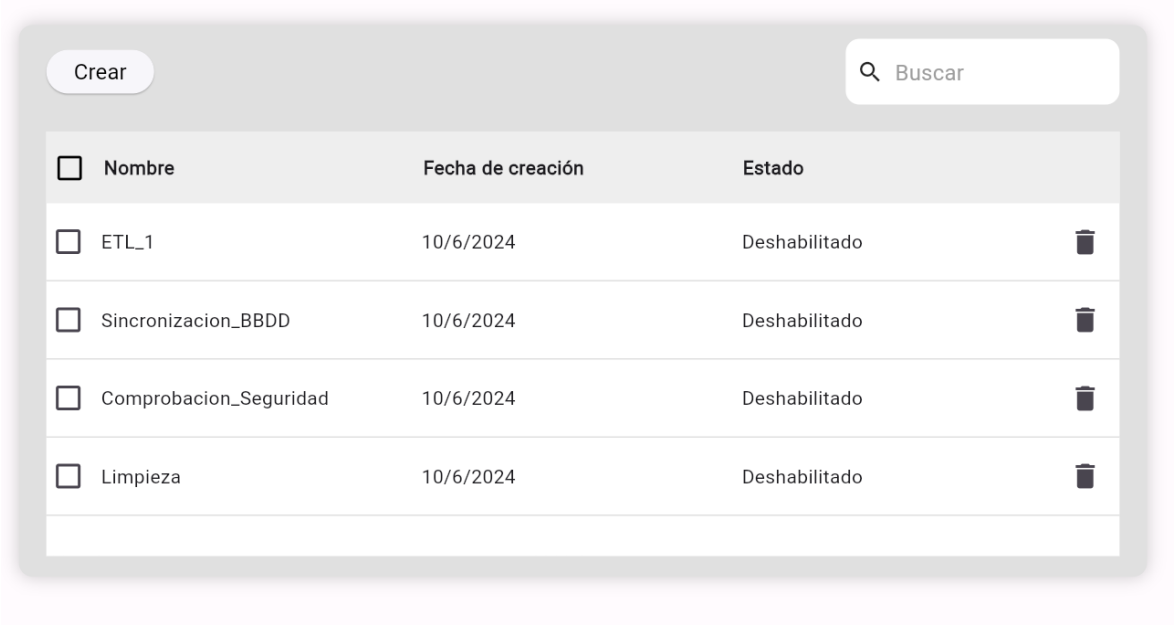

Figura 6.2: Pestaña Tareas

Al acceder a la interfaz para la creación de la tarea, nos encontramos con distintos apartados a rellenar para completar los datos de la tarea. En primer lugar, se debe rellenar la información general de la tarea como son el nombre, la descripción o el estado de la tarea (figura 6.3).

Si se continúa, se define la información de destino de la tarea (figura 6.4), debiendo rellenar la url a la que se hace la petición, el método HTTP y el body de la petición.

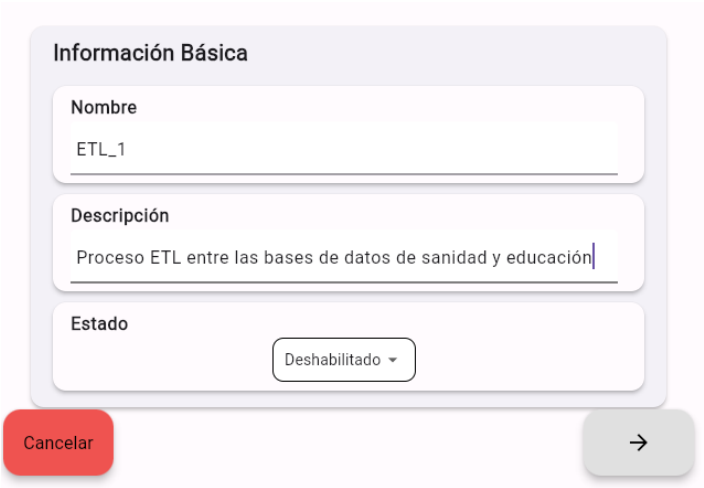

Figura 6.3: Información General de la tarea

Figura 6.4: Información de Destino de la tarea

Por último, se debe rellenar la información de la programación de la tarea. En esta pestaña existen tres modos.

- La programación única de la tarea (figura 6.5). Se debe indicar obligatoriamente la fecha y la hora de ejecución.
- La programación en modo rate (figura 6.6). Se basa en la frecuencia y se debe elegir el valor y la unidad (minutos, horas o días).
- La programación en modo cron (figura 6.7). Se debe indicar la expresión de cron rellenando cada uno de los campos.

Tanto en el modo rate como en cron se puede definir una fecha de inicio y de fin para la ejecución de la tarea. Además, mediante la declaración de la zona horaria, se puede ajustar a la localización del usuario para evitar conflictos entre zonas.

Figura 6.5: Programación única de la tarea

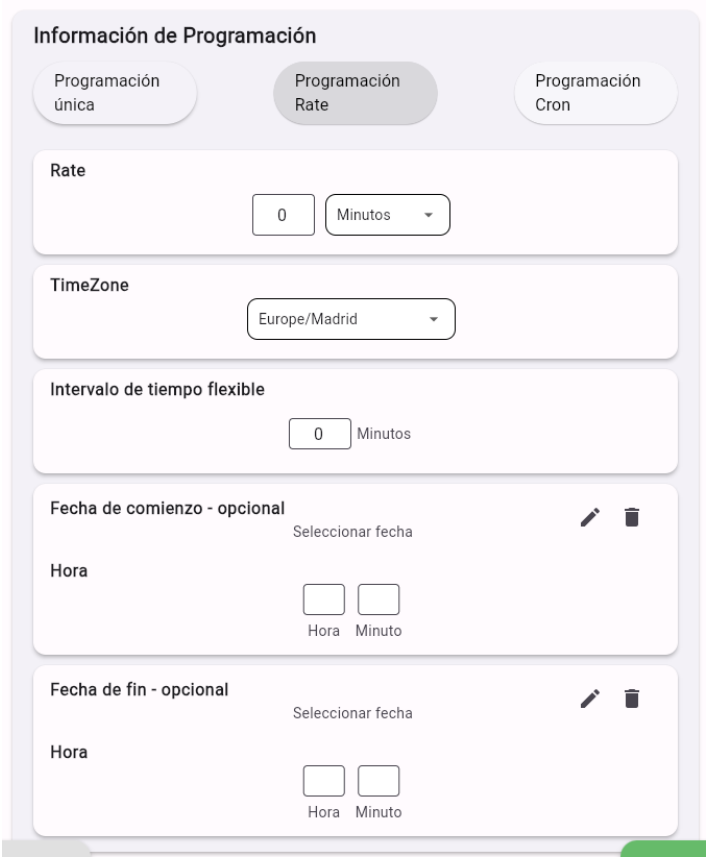

Figura 6.6: Programación Rate de la tarea

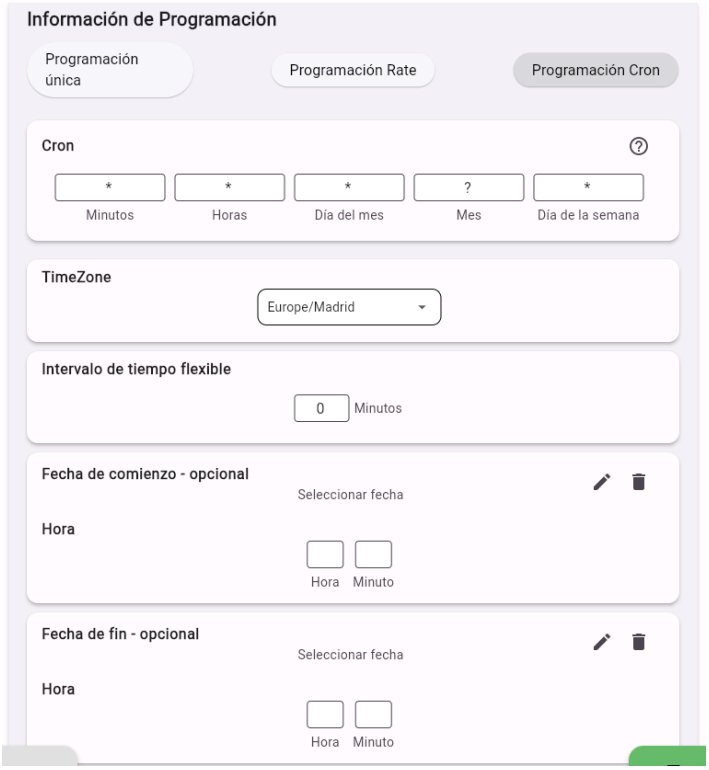

Figura 6.7: Programación Cron de la tarea

Si se opta por la programación de la tarea con cron, existe un un apartado de ayuda, si se accede a este desplegable (figura 6.8), podrás utilizar el asistente virtual con el que se puede obtener la expresión de cron necesaria para la tarea. Para ello, deberás preguntarle y obtendrás la respuesta con la expresión. Además, se pueden comprobar las futuras 5 ejecuciones para ver que es correcta. Por último, si se quiere utilizar esta expresión, se puede pulsando el botón ”Utilizar expresión”.

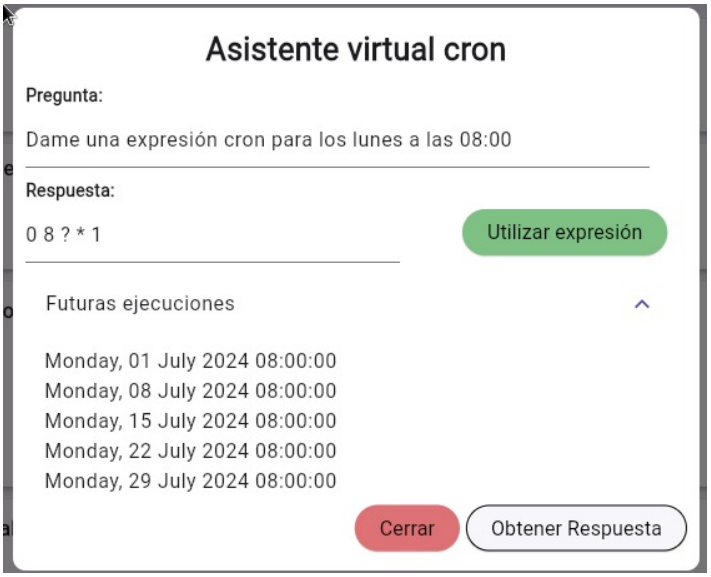

Figura 6.8: Asistente inteligente de cron

Una vez creada la tarea, si se quiere modificar los datos de la tarea, basta con pulsar la tarea en la lista de tareas del usuario (figura 6.2) para acceder a la interfaz donde podrás modificar sus datos, en esta interfaz aparecen los mismos cuadros que en las figuras 6.3, 6.4, 6.5, 6.6 y 6.7.

La única diferencia, es que se muestran tres botones (figura 6.9), un botón para eliminar la tarea, un segundo botón para guardar la tarea y además un botón que muestra un desplegable donde se muestran los eventos generados por la tarea cada vez que se cumplía con la planificación que detalla el usuario (figura 6.10).

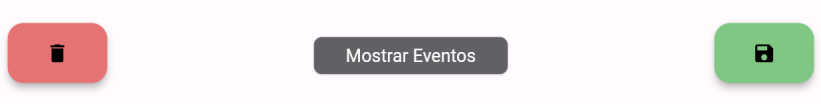

Figura 6.9: Botones finales de la pestaña Tarea

Figura 6.10: Eventos generados por la tarea

6.2 Coste del Proyecto

En este apartado se presenta un desglose detallado del coste del proyecto para la construcción de una aplicación destinada a la programación de tareas en entornos de computación en la nube.

El principal coste del proyecto corresponde al salario del programador encargado del desarrollo de la aplicación. Se ha utilizado el salario bruto proporcionado por la empresa al estudiante para calcular este coste.

Las herramientas de programación empleadas en el proyecto han sido gratuitas o proporcionadas a través de licencias otorgadas por la universidad o la empresa, por lo que no han representado un coste adicional significativo.

El hardware utilizado durante el desarrollo fue un portátil proporcionado por la empresa. Para calcular su coste, se ha incluido el coste de amortización del equipo.

Se ha estimado el coste del despliegue de la aplicación en la nube durante un año. Se ha utilizado AWS Kubernetes para el despliegue con un coste de 0,1 $/hora. Además, se ha considerado el precio de la compra de un dominio para alojar la aplicación en Internet, que asciende a 10 $/año. Por último, la utilización de una base de datos de respaldo para la aplicación con AWS S3, estimando el coste por el almacenamiento que se necesitará para la aplicación y las llamadas a la API para modificar estos datos.

El tiempo de desarrollo del proyecto ha sido de 4 meses y medio, abarcando desde mediados de febrero de 2024 hasta junio de 2024.

En la tabla 6.3 se muestra el desglose detallado de los costes del proyecto, con un coste total de 6.943 €.

Cuadro 6.3: Tabla de precios del proyecto

Recurso	**Precio**	**Duración**	**Precio**
Salario del programador	1300 €/mes	4,5 meses	5.850 €
Amortización portátil de la empresa	30 €/mes	4,5 meses	135 €
Despliegue en la nube	0.1€/hora	1 año	876 €
Backup almacenamiento de datos	3 €/mes	1 año	36 €
Consultas a la base de datos	3 €/mes	1 año	36 €
Dominio web	10 €/año	1 año	10 €
		Total:	**6943 €**

Capítulo 7

Conclusiones

PARA finalizar, en este capítulo se discutirá el grado de consecución de los objetivos del proyecto. También se plasmarán las competencias de la intensificación asociadas al mismo. A continuación, se dará una opinión personal tras la realización del presente TFG. Por último, se presentarán diversas ideas con las que se podría trabajar en el futuro para la mejora de la aplicación. Con este capítulo se concluye la memoria de este TFG.

7.1 Objetivos alcanzados

- **Estudio comparativo de las principales plataformas de computación en la nube y del soporte que ofrecen para tareas en segundo plano.** Este objetivo ha sido cumplido. Primero, se ha hecho un estudio comparativo, resultando en que todas tenían un rendimiento bastante similar, por lo que se ha hecho un estudio en profundidad de los costes de la programación de las tareas. Con este objetivo, se ha tomado una gran base de conocimiento sobre las plataformas.
- **Diseño y desarrollo del microservicio que permita la programación de tareas.** Este objetivo ha sido cumplido. Se ha desarrollado un microservicio que, abstrayendo al usuario de la plataforma de computación en la nube, le permite crear tareas en segundo plano. La plataforma elegida ha sido AWS.
- **Diseño y desarrollo de los microservicios auxiliares.** Se han desarrollado tres microservicios auxiliares que le permiten al usuario una experiencia más agradable al interactuar con la aplicación. Por un lado se ha creado un microservicio con la interfaz para la interacción del usuario con la aplicación. Por otro lado se ha creado un microservicio de gestión de los datos de los usuarios. El último microservicio se encarga de mostrar los eventos generados por las tareas en segundo plano.
- **Desarrollo de un módulo de apoyo a la toma de decisiones relativo al perfilado de horarios.** Tras barajar varias alternativas para la ayuda al usuario, se ha optado por el entrenamiento de un LLM creando así un asistente virtual. Con este asistente virtual, se le facilita la tarea al usuario a la hora de crear tareas cron, pues se puede interactuar mediante lenguaje natural para obtener expresiones cron, las cuales tienen una sintaxis que puede ser complejas de entender para el usuario.

- **Despliegue de los microservicios.** Aunque este objetivo ha planteado cierto reto, se ha completado de manera correcta. Para facilitar la transición entre la fase de desarrollo y de despliegue se han empaquetado los microservicios en contenedores y se han almacenado en un repositorio. Además, para el despliegue se ha utilizado K8s facilitando el despliegue en local. Por último, se ha expuesto la aplicación a Internet mediante ngrok en la dirección "adapted-first-snipe.ngrok-free.app". La complicación encontrada reside en la posibilidad de utilizar un Virtual Private Server (VPS) para el despliegue de la aplicación, sin embargo, no se han encontrado VPS asequibles que cumplan con los requisitos técnicos de esta aplicación.
- **Testing y validación de la aplicación.** Este objetivo se ha cumplido. Primero, se han realizado pruebas unitarias a los diversos microservicios de backend. A continuación, se ha probado el asistente virtual mediante un conjunto de datos de prueba con los que se han conseguido las salidas esperadas por el asistente. Por último, se ha valid ado la interfaz de la aplicación con el tutor de la empresa, mejorando dicha interfaz.

7.2 Competencias cumplidas

- **[CM4] Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.** Durante la fase de diseño, se exploraron diversas alternativas para optimizar la gestión de tareas en la nube. Se encontraron limitaciones técnicas que llevaron a descartar algunas opciones. La falta de flexibilidad para modificar el mecanismo de programación de las tareas en las plataformas de la nube descartó la posibilidad de implementar este sistema. Se realizó un estudio para el desarrollo de un Decision Support System (DSS) como marco de aplicación para la programación de tareas y también se consideró el DSS como sistema de apoyo, aunque se prefirió una alternativa más adecuada para el proyecto. Sin embargo, el proceso proporcionó un profundo conocimiento de los sistemas inteligentes. Este proceso amplió la comprensión teórica y facilitó la toma de decisiones informadas para garantizar la efectividad del sistema seleccionado.
- **[CM7] Capacidad para conocer y desarrollar técnicas de aprendizaje computacional y diseñar e implementar aplicaciones y sistemas que las utilicen, incluyendo las dedicadas a extracción automática de información y conocimiento a partir de grandes volúmenes de datos.** Una de las características clave de esta aplicación es la integración de técnicas de aprendizaje computacional. Específicamente, se ha adquirido esta competencia en el contexto del entrenamiento y ajuste de modelos de aprendizaje automático. Como parte de este desarrollo, se ha realizado el *fine-tuning* de un LLM para crear un asistente inteligente. En este sentido, ha sido necesario estu-

diar este tipo de modelos y cómo funcionan. Otras alternativas como el uso de redes neuronales o de algoritmos de aprendizaje automático también han sido valoradas y estudiadas, reforzando la consecución de esta competencia.

7.3 Opinión personal

Este proyecto marca mi final en el Grado en Ingeniería Informática. Durante estos cuatro años he aprendido una gran variedad de conocimientos en la escuela, proporcionándome la base para el desarrollo de esta aplicación. Si bien es cierto, durante el desarrollo de la aplicación se han aprendido nuevas tecnologías y frameworks que han sido y serán de utilidad para futuros proyectos.

Este trabajo ha sido un reto tanto personal como profesional, el desarrollo de una aplicación, dentro de una empresa pero trabajando de manera individual. En este desarrollo se ha tratado de asemejar lo más fiel posible a como sería el desarrollo en un equipo de trabajo con distintos compañeros, por lo que ha sido un quebradero de cabezas que me ha permitido aprender la atención que se debe dar al más mínimo detalle, especialmente en las primeras etapas, donde una buena planificación y diseños permite aprovechar mejor el tiempo.

7.4 Trabajo futuro

- **Entreno del asistente virtual con más datos**. Aunque el asistente virtual hasta este punto es robusto en cuanto al entendimiento y generación de la sintaxis de cron, se debería crear un conjunto mayor de datos con el que entrenar el modelo y hacer que sea más fiable cuando se trata de expresiones complejas que requieran diversos operadores para su definición. Además, sería una gran ayuda si el asistente de cron tuviera una visión general de la aplicación y fuera capaz de ofrecer ayuda en cualquier momento, por ejemplo mostrarle al usuario el objetivo de la aplicación, en que pestaña encontrar el elemento que busca y detallar la función de cada campo a rellenar en las tareas.
- **Selección de la plataforma de computación en la nube**. Solamente se ha seleccionado AWS como plataforma para la creación de tareas en la nube. Si bien esta solución es ideal para la mayoría de usuarios, es posible que alguno de ellos prefiera utilizar una plataforma para la creación de tareas por preferencia personal. Añadir una opción que te permita definir en que plataforma defines la tarea sería un gran añadido a la aplicación.
- **Despliegue de la aplicación en producción**. Se ha desplegado la aplicación en una plataforma de Kubernetes en local y aunque se puede acceder como en un entorno en producción, un pequeño paso más de exportar la definición del cluster local a un servicio en la nube como Amazon Elastic Kubernetes Service o Google Kubernetes Engine mejoraría la eficiencia y daría más recursos de computación a la aplicación.

Anexos

Anexo A

A.1 Despliegue de la aplicación

Este anexo detalla el proceso para desplegar los microservicios y exponer la aplicación a través de Internet. Es importante destacar que todos los archivos mencionados están disponibles en el repositorio de despliegue de la aplicación. Sin embargo, no se incluyen dos archivos cruciales para el despliegue: el archivo de secretos, necesario para mantener la seguridad de los componentes de la aplicación. Y el certificado con el que se despliegan los microservicios. En caso de ser necesario, se puede contactar con el alumno para obtener los archivos.

- `https://github.com/Forte-GestorJobs/despliegue`

El siguiente tutorial descrito se ha realizado en Ubuntu 22.04 y ha sido necesario instalar 3 aplicaciones.

- Kubernetes[1]
- Nginx[2]
- Ngrok[3]

A.1.1 Despliegue con Kubernetes

En el repositorio existe un Makefile con el que se facilita el despliegue de los microservicios. Dentro del mismo repositorio hay una serie de archivos con la extensión yaml con los que se despliega en K8s la aplicación. Será tan sencillo como ejecutar el Makefile (código A.1) o ejecutar cada uno de los comandos del Makefile por separado (código A.2)

```
1 make start-k8s
```

Code listing A.1: Makefile

```
1         kubectl apply -f persistent-volumes.yaml
2         kubectl apply -f bbdds.yaml
```

[1] `https://docs.k3s.io/`
[2] `https://nginx.org/en/`
[3] `https://ngrok.com/`

```
3           kubectl apply -f micro-cloud-eventos.yaml
4           kubectl apply -f micro-sistema-apoyo.yaml
5           kubectl apply -f micro-frontend-main.yaml
6           kubectl apply -f micro-cloud-tareas.yaml
7           kubectl apply -f micro-gestion-usuarios.yaml
```

Code listing A.2: Despliegue de los microservicios

A.1.2 Despliegue con Nginx

Una vez se han desplegado los microservicios en local con Kubernetes, hay que utilizar un reverse proxy para unificar todos los puertos en uno solo y así poder exponer la aplicación con ngrok en el siguiente paso.

Para ello, una vez instalado nginx, se deberá crear un archivo en la carpeta "/etc/nginx/sites-available/" con el nombre nginx.conf (El directorio puede variar dependiendo del sistema).

El archivo nginx.conf tendrá el siguiente contenido (código A.3).

```
1  server {
2      listen 8443;
3      server_name localhost;

5      ssl_certificate /etc/nginx/ssl/micros-gestor-jobs.pem;
6      ssl_certificate_key /etc/nginx/ssl/micros-gestor-jobs.pem;
7      location / {
8          proxy_pass https://localhost:30812/;
9          proxy_set_header Host $host;
10         proxy_set_header X-Real-IP $remote_addr;
11         proxy_set_header X-Forwarded-For
               $proxy_add_x_forwarded_for;
12     }

14     location /api/user/ {
15         proxy_pass https://localhost:30808/user/;
16         proxy_set_header Host $host;
17         proxy_set_header X-Real-IP $remote_addr;
18         proxy_set_header X-Forwarded-For
               $proxy_add_x_forwarded_for;
19     }
20     location /api/task/ {
21         proxy_pass https://localhost:30809/task/;
22         proxy_set_header Host $host;
```

```
23          proxy_set_header X-Real-IP $remote_addr;
24          proxy_set_header X-Forwarded-For
               $proxy_add_x_forwarded_for;
25      }
26      location /api/event/ {
27          proxy_pass https://localhost:30810/event/;
28          proxy_set_header Host $host;
29          proxy_set_header X-Real-IP $remote_addr;
30          proxy_set_header X-Forwarded-For
               $proxy_add_x_forwarded_for;
31      }
32      location /api/chatbot {
33          proxy_pass https://localhost:30811/chatbot;
34          proxy_set_header Host $host;
35          proxy_set_header X-Real-IP $remote_addr;
36          proxy_set_header X-Forwarded-For
               $proxy_add_x_forwarded_for;
37      }

39  }
```

Code listing A.3: Configuración de Nginx

A.1.3 Exposición con Ngrok

En primer lugar será necesario registrarse en Ngrok. A continuación, se puede seguir el tutorial que ofrece Ngrok para la instalación en su página principal. Cabe recalcar que se debe configurar el token de acceso.

A continuación, se debe crear un dominio para la aplicación, accediendo al panel ”Domains” dentro de la página (figura A.1).

Por último, con el dominio generado, se despliega la aplicación de la siguiente manera, sustituyendo el dominio por el generado en la aplicación (código A.4).

```
1  ngrok http http://localhost:8443 --domain=adapted-first-snipe.
       ngrok-free.app
```

Code listing A.4: Ngrok

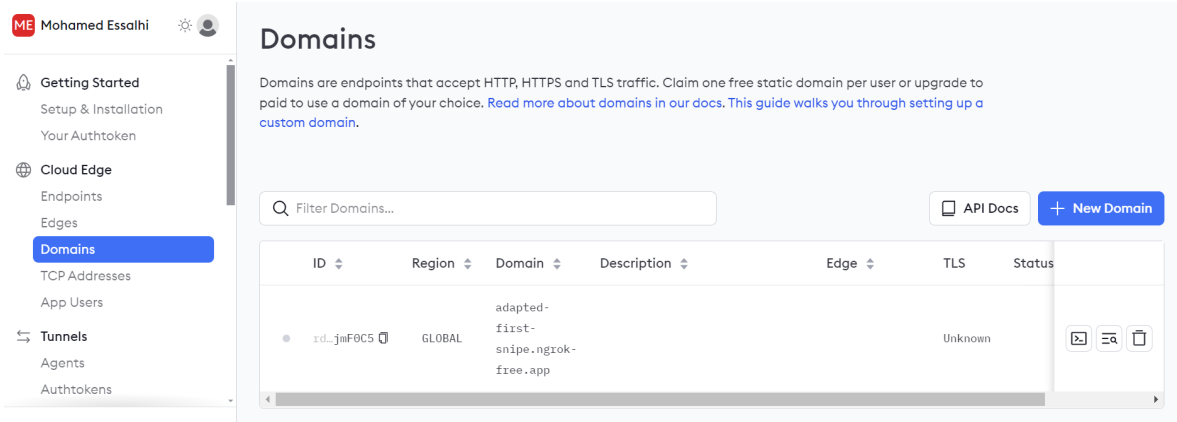

Figura A.1: Creación del dominio en ngrok

Referencias

[AAG+14] Hussain AlJahdali, Abdulaziz Albatli, Peter Garraghan, Paul Townend, Lydia Lau, y Jie Xu. Multi-tenancy in Cloud Computing. En *2014 IEEE 8th International Symposium on Service Oriented System Engineering*, páginas 344–351, 2014.

[AEB16] Meryeme Alouane y Hanan El Bakkali. Virtualization in Cloud Computing: Existing solutions and new approach. En *2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*, páginas 116–123, 2016.

[AKM18] Guruh Aryotejo, Daniel Y Kristiyanto, y Mufadhol. Hybrid cloud: bridging of private and public cloud computing. *Journal of Physics: Conference Series*, 1025(1):012091, may 2018. url: https://dx.doi.org/10.1088/1742-6596/1025/1/012091.

[Ama23] T. Amaratunga. *Understanding Large Language Models: Learning Their Underlying Concepts and Technologies*. Apress, 2023. url: https://books.google.es/books?id=cyQo0AEACAAJ.

[And15] Charles Anderson. Docker [Software engineering]. *IEEE Software*, 32(3):102–c3, 2015.

[Awa05] MA Awad. A comparison between agile and traditional software development methodologies. *University of Western Australia*, 30:1–69, 2005.

[BH08] F. Burstein y C.W. Holsapple. *Handbook on Decision Support Systems 1: Basic Themes*. International Handbooks on Information Systems. Springer Berlin Heidelberg, 2008. url: https://books.google.es/books?id=q_3sRkRKZQwC.

[BHA+21] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ B. Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo

Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen Creel, Jared Quincy Davis, Dorottya Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, y et al. On the Opportunities and Risks of Foundation Models. *CoRR*, abs/2108.07258, 2021. url: `https://arxiv.org/abs/2108.07258`.

[BHW81] Robert H. Bonczek, Clyde W. Holsapple, y Andrew B. Whinston. *Foundations of Decision Support Systems*. Number 9780121130503 in Elsevier Monographs. Elsevier, 1981. url: `https://ideas.repec.org/b/eee/monogr/9780121130503.html`.

[Bid96] Hossein Bidgoli. A new productivity tool for the 90's: Group support systems. *Journal of Systems management*, 47(4):56, 1996.

[Buc09] *Model-View-Controller Pattern*, páginas 353–402. Apress, Berkeley, CA, 2009. url: `https://doi.org/10.1007/978-1-4302-2370-2_20`.

[CA07] Swarat Chaudhuri y Rajeev Alur. Instrumenting C Programs with Nested Word Monitors. En Dragan Bošnački y Stefan Edelkamp, editors, *Model Checking Software*, páginas 279–283, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[CGB+21] Benjamin Clavié, Akshita Gheewala, Paul Briton, Marc Alphonsus, Rym Laabiyad, y Francesco Piccoli. LegaLMFiT: Efficient Short Legal Text Classification with LSTM Language Model Pre-Training. *CoRR*, abs/2109.00993, 2021. url: `https://arxiv.org/abs/2109.00993`.

[DL19] Lorenzo De Lauretis. From Monolithic Architecture to Microservices Architecture. En *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, páginas 93–96, 2019.

[Fre15] Adam Freeman. *The Builder Pattern*, páginas 233–250. Apress, Berkeley, CA, 2015. url: `https://doi.org/10.1007/978-1-4842-0394-1_11`.

[GZ20] Konrad Gos y Wojciech Zabierowski. The Comparison of Microservice and Monolithic Architecture. En *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, páginas 150–153, 2020.

[HBB17] K. Hightower, B. Burns, y J. Beda. *Kubernetes: Up and Running: Dive into the Future of Infrastructure*. O'Reilly Media, 2017. url: https://books.google.es/books?id=fF4KswEACAAJ.

[Hol03] C. Holsapple. *Handbook on Knowledge Management 1: Knowledge Matters*. Handbook on Knowledge Management. Springer, 2003. url: https://books.google.es/books?id=VHf7erhkZfgC.

[IKKO14] Hiroshi Inamura, Takeshi Kamiyama, Teppei Konishi, y Ken Ohta. Extending Battery Lifetime in Smartphones with Power Efficient Task Management and Energy Aware Design Tool. *International Journal of Informatics Society*, 6(1):3–10, 2014.

[JSM+23] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, y William El Sayed. Mistral 7B, 2023.

[JVS21] T. Jaskula, V. Vernon, y an O'Reilly Media Company Safari. *Strategic Monoliths and Microservices: Driving Innovation Using Purposeful Architecture*. Addison-Wesley Professional, 2021. url: https://books.google.es/books?id=q4mgzgEACAAJ.

[KMA19] Mikael Koskinen, Tommi Mikkonen, y Pekka Abrahamsson. *Containers in Software Development: A Systematic Mapping Study*, páginas 176–191. 11 2019.

[KS22] Sandra Kublik y Shubham Saboo. *GPT-3*. O'Reilly Media, Incorporated, 2022.

[LE93] Dorothy E. Leidner y Joyce J. Elam. Executive Information Systems: Their Impact on Executive Decision Making. *Journal of Management Information Systems*, 10(3):139–155, Diciembre 1993. url: https://doi.org/10.1080/07421222.1993.11518014.

[LHL+20] Guozhi Liu, Bi Huang, Zhihong Liang, Minmin Qin, Hua Zhou, y Zhang Li. Microservices: architecture, container, and challenges. En *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, páginas 629–635, 2020.

[LL09] Kathryn B. Laskey y Kenneth Laskey. Service oriented architecture. *WIREs Computational Statistics*, 1(1):101–105, 2009. url: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wics.8.

[LL21] Robert La Lau. *Task Scheduling*, páginas 175–181. Apress, Berkeley, CA, 2021. url: https://doi.org/10.1007/978-1-4842-6960-2_8.

[Mar08] R.C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Robert C. Martin Series. Pearson Education, 2008. url: https://books.google.es/books?id=_i6bDeoCQzsC.

[MLB+11] Sean Marston, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang, y Anand Ghalsasi. Cloud computing — The business perspective. *Decision Support Systems*, 51(1):176–189, 2011. url: https://www.sciencedirect.com/science/article/pii/S0167923610002393.

[MLS20] R. Modi, J. Lee, y R. Skaria. *Azure for Architects: Create secure, scalable, high-availability applications on the cloud, 3rd Edition*. Packt Publishing, 2020. url: https://books.google.es/books?id=3TfyDwAAQBAJ.

[Mus23] O. Musch. *Design Patterns with Java: An Introduction*. Springer Fachmedien Wiesbaden, 2023. url: https://books.google.es/books?id=kQmtEAAAQBAJ.

[MV23] H.P. Martinez y I.H. Vargas. *Google Cloud for Developers: Write, migrate, and extend your code by leveraging Google Cloud*. Packt Publishing, 2023. url: https://books.google.es/books?id=3V3DEAAAQBAJ.

[New15] Sam Newman. *Building Microservices*. O'Reilly Media, Inc., edición 1st, 2015.

[NMMA16] I. Nadareishvili, R. Mitra, M. McLarty, y M. Amundsen. *Microservice Architecture: Aligning Principles, Practices, and Culture*. O'Reilly Media, Incorporated, 2016. url: https://books.google.es/books?id=BvUEvgAACAAJ.

[PC20] Ben Piper y David Clinton. *AWS Certified Solutions Architect Study Guide Third Edition*. Sybex, 2020.

[PMT21] Severi Peltonen, Luca Mezzalira, y Davide Taibi. Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review. *Information and Software Technology*, 136:106571, 2021. url: https://www.sciencedirect.com/science/article/pii/S0950584921000549.

[QLDG09] Ling Qian, Zhiguo Luo, Yujian Du, y Leitao Guo. Cloud Computing: An Overview. En Martin Gilje Jaatun, Gansen Zhao, y Chunming Rong, editors, *Cloud Computing*, páginas 626–631, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[Ric15] Mark Richards. *Microservices vs. service-oriented architecture*. O'Reilly Media Sebastopol, 2015.

[Ric18] C. Richardson. *Microservices Patterns: With examples in Java*. Manning, 2018. url: https://books.google.es/books?id=UeK1swEACAAJ.

[RNM06] William Richmond, Paul Nelson, y Sanjog Misra. An Empirical Analysis of Software Life Spans to Determine the Planning Horizon for New Software. *Information Technology and Management*, 7:131–149, 04 2006.

[Rub12] K.S. Rubin. *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Addison-Wesley signature series. Addison-Wesley, 2012. url: https://books.google.es/books?id=HkXX65VCZU4C.

[Sar16] Vaskaran Sarcar. *Adapter Patterns*, páginas 47–52. Apress, Berkeley, CA, 2016. url: https://doi.org/10.1007/978-1-4842-1802-0_8.

[SGP19] C. Surianarayanan, G. Ganapathy, y R. Pethuru. *Essentials of Microservices Architecture: Paradigms, Applications, and Techniques*. CRC Press LLC, 2019. url: https://books.google.es/books?id=0iX8zwEACAAJ.

[Sim60] H.A. Simon. *The New Science of Management Decision*. Ford distinguished lectures. Harper, 1960. url: https://books.google.es/books?id=nktqAAAAMAAJ.

[ST20] Manish Saraswat y R.C. Tripathi. Cloud Computing: Analysis of Top 5 CSPs in SaaS, PaaS and IaaS Platforms. En *2020 9th International Conference System Modeling and Advancement in Research Trends (SMART)*, páginas 300–305, 2020.

[SvD19] M. Seemann y S. van Deursen. *Dependency Injection Principles, Practices, and Patterns*. Manning, 2019. url: https://books.google.es/books?id=zzczEAAAQBAJ.

[SW07] James Shore y Shane Warden. *The art of agile development*. O'Reilly, edición First, 2007.

[Uza22] S.B. Uzayr. *Software Design Patterns: The Ultimate Guide*. Ultimate Guide. CRC Press, 2022. url: https://books.google.es/books?id=Ses5zwEACAAJ.

[VSP+23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, y Illia Polosukhin. Attention Is All You Need, 2023.

[WB10] Yi Wei y M. Brian Blake. Service-Oriented Computing and Cloud Computing: Challenges and Opportunities. *IEEE Internet Computing*, 14(6):72–75, 2010.

[WDC+21] Siao Wang, Chenglie Du, Jinchao Chen, Ying Zhang, y Mei Yang. Microservice Architecture for Embedded Systems. En *2021 IEEE 5th Information Technology,Networking,Electronic and Automation Control Conference (ITNEC)*, volume 5, páginas 544–549, 2021.

[XZ12] Yuping Xing y Yongzhao Zhan. Virtualization and Cloud Computing. En Ying Zhang, editor, *Future Wireless Networks and Information Systems*, páginas 305–312, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[YDM02] WS Yu, DA Doroja, y JM Monje. *Developing A UTC-synchronized University Network Time Service*. PhD thesis, Citeseer, 2002.

Este documento fue editado y tipografiado con LaTeX empleando
la clase **esi-tfg** (versión 0.20181017) que se puede encontrar en:
`https://bitbucket.org/esi_atc/esi-tfg`