Universitat Oberta de Catalunya (UOC)

Máster Universitario en Ciencia de Datos (*Data Science*)

TRABAJO FINAL DE MÁSTER

Ciclo de Vida de los Datos y MLOps: Herramientas, metodologías, puesta en producción y mantenimiento de proyectos de Data Science.

Autor: David Herrero Pascual

Tutor: Iván González Torre

Coordinador: Albert Solé Ribalta

Madrid, 5 de junio de 2022

Créditos/Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	Ciclo de Vida de los Datos y MLOps: Herramientas, metodologías, puesta en producción y mantenimiento de proyectos de Data Science.
Nombre del autor:	David Herrero Pascual
Nombre del colaborador/a docente:	Iván González Torre
Nombre del PRA:	Albert Solé Ribalta
Fecha de entrega (mm/aaaa):	06/2022
Titulación o programa:	Máster Universitario en Ciencia de Datos (*Data Science*)
Área del Trabajo Final:	Área 4, temática libre: MLOps
Idioma del trabajo:	Español
Palabras clave	Data Lifecycle, MLOps, Data Science, DevOps, Artificial Intelligence

Cita

“La herramienta más poderosa que tenemos como desarrolladores es la automatización¨

- Scott Hanselman,

Abstract - Castellano

En este proyecto, se aborda la problemática de MLOps, es decir, técnicas DevOps aplicadas a proyectos de Inteligencia Artificial. El trabajo se ha escrito de manera que sirva como guía a la hora de abordar la implantación de técnicas y herramientas MLOps a cualquier escala, y a lo largo de los capítulos se van desarrollando diferentes ideas que ayudan al lector a comprender qué es MLOps, de dónde surge, por qué, y cómo se implementa, para llegar en última instancia a la implementación de un sistema de demostración.

Concretamente, tras un capítulo introductorio, el capítulo 2 expone el estado del arte de metodologías y tecnologías DevOps y MLOps, para que a continuación, en el capítulo 3, se expongan herramientas que permitan la implementación de una metodología eficaz. Por último, en el capítulo 4, se desarrolla una arquitectura de prueba para demostrar las capacidades de algunas de estas herramientas, generando un pipeline automatizado y con trazabilidad completa desde el principio del desarrollo hasta la puesta en producción.

Abstract - English

This project addresses the issue of MLOps, i.e. DevOps techniques applied to Artificial Intelligence projects. The work has been written to serve as a guide for the implementation of MLOps techniques and tools at any scale, and throughout the chapters different ideas are developed to help the reader understand what MLOps is, where it comes from, why, and how it is implemented, to ultimately arrive at the implementation of a demonstration system.

Specifically, after an introductory chapter, Chapter 2 presents the state of the art of DevOps and MLOps methodologies and technologies, followed by Chapter 3, which presents tools that enable the implementation of an effective methodology. Finally, in chapter 4, a test architecture is developed to demonstrate the capabilities of some of these tools, generating an automated pipeline with complete traceability from the beginning of development to production.

Palabras clave: Data Lifecycle, MLOps, Data Science, DevOps, Artificial Intelligence.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Descripción General del Problema

A lo largo de la última década, y especialmente en los últimos años, el número de proyectos de inteligencia artificial que se desarrollan ha crecido exponencialmente, así como la inversión para desarrollar proyectos y nuevas técnicas de IA. Según datos de Statista, en 2020 la inversión mundial en Inteligencia Artificial rondó los 68.000 millones de dólares frente a los 44.000 millones en 2017 [3]. Esto supone un crecimiento de 24.000 millones de dólares en solamente 3 años.

Un crecimiento y una inversión tan fuerte implica también una serie de problemas que no se deben dejar de lado, especialmente aquellos que tienen que ver con la puesta en producción de los modelos desarrollados. Según Gartner, la principal consultora de investigación de las TIC, se estima que el 85 % de los proyectos de Inteligencia Artificial fallarán y arrojarán resultados erróneos hasta el año 2022 [4]. Además, el 70 % de las compañías reportan un impacto mínimo de la IA en sus organizaciones [5], y un 87 % de los proyectos de Data Science nunca llegan a producción [6].

Con estos datos en la mano, resulta vital tratar de identificar todos los problemas que surgen a la hora de poner en producción proyectos de Ciencia de Datos, ya que una inversión tan grande no debería quedarse en una simple prueba de concepto o en un algoritmo del que no se pueda sacar valor a nivel de negocio.

Este Trabajo de Fin de Máster tratará de identificar toda la problemática derivada de poner en producción proyectos de Data Science, así como de exponer las diferentes metodologías y herramientas existentes en el ámbito de MLOps. Una vez identificado el problema y las

posibles soluciones, se desarrollará e implementará un caso de uso que demuestre la utilidad de las metodologías MLOPs. Aunque no es el objetivo principal del TFM, el caso de uso sirve como base para cualquier arquitectura MLOPs, ya sea a pequeña o gran escala. Por tanto, los objetivos del proyecto son:

- Desarrollar una guía de desarrollo software orientado a Data Science.
- Desarrollar el estado del arte de las metodologías y herramientas MLOps.
- Describir metodologías, herramientas y técnicas de DevOps aplicado a Inteligencia Artificial (MLOps)
- Desarrollar una comparativa de herramientas para el caso de uso planteado.
- Diseñar e implementar una arquitectura de MLOps, capaz de:
  - Llevar una trazabilidad de los experimentos (modelos y datos utilizados).
  - Automatizar la subida a producción de las actualizaciones del modelo.
  - Demostrar las ventajas de la automatización a la hora de poner en producción un modelo de Inteligencia Artificial.

En definitiva, desarrollar una guía de cómo evitar que los proyectos de Data Science se queden en desarrollo y nunca lleguen a producción y ejemplificar un caso de éxito.

1.2. Motivación y justificación del proyecto

1.2.1. Motivación personal

Respecto a mi motivación personal para la realización de este proyecto, hay varios puntos que me gustaría comentar.

En primer lugar, actualmente trabajo en el área de Inteligencia Artificial de una empresa, y cada día veo la dificultad que tiene definir una metodología común a todos los integrantes de la unidad y sobretodo lo complicado que es en tiempo y esfuerzo el llevar los proyectos a producción. Por eso, considero vital que se desarrollen estas herramientas y técnicas de despliegue e integración continua en proyectos de Ciencia de Datos.

Además, en muchas ocasiones me he encontrado con clientes algo reticentes a invertir dinero para proyectos de Inteligencia Artificial precisamente porque piensan que es una inversión arriesgada y que luego existe la probabilidad de que el proyecto quede en un I+D y no en un proyecto productivo que les aporte valor en el día a día.

Por todos esos motivos, creo que es importante que se desarrollen las técnicas de implementación y puesta en producción de los proyectos de ciencia de datos a la vez que se desarrollan también las propias técnicas para desarrollar estos proyectos, de forma que no solo los resultados de los modelos y algoritmos sean mejores, sino que además sean más fáciles de poner en marcha y que los clientes empiecen a recuperar la inversión realizada lo antes posible.

1.2.2. Metodología y plan de investigación

Para la realización de este proyecto se seguirá una metodología basada en entregas periódicas, con feedback constante por parte del tutor. Estas entregas se distribuyen en el siguiente diagrama de Gantt.

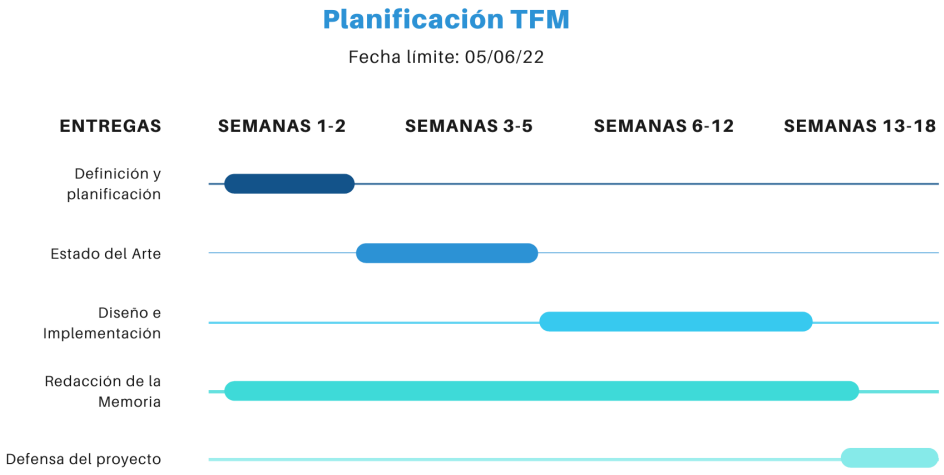

Figura 1.1: Diagrama de Gantt del proyecto

Como se puede observar en el diagrama, la redacción de la memoria se irá haciendo de forma constante a lo largo de todo el desarrollo. Una vez terminado el desarrollo y la memoria, se realizará una presentación y defensa del proyecto.

Con respecto a la investigación, se dividirá en dos partes: Tecnologías y Datos.

1.2.2.1. Tecnologías

Las tecnologías utilizadas, su historia y sus usos, así como el estado del arte actual en este tipo de proyectos se describirán en dos secciones distintas. En la primera sección se describirán las tecnologías utilizadas en el proyecto, así como las diferentes alternativas tecnológicas que se han descartado.

Por otro lado, en la sección de estado del arte, se describirán herramientas y otras soluciones ya desarrolladas, así como artículos científicos y documentos de investigación.

El objetivo de estas dos secciones es justificar el sentido de este proyecto, la innovación que supone y el por qué de que se hayan usado unas tecnologías y no otras.

1.2.2.2. Datos

Para el desarrollo del proyecto se ha utilizado el dataset MNIST, disponible desde la API de Keras. La obtención de los datos se detalla en el anexo A

Capítulo 2

Estado del Arte

En esta sección se expone el estado actual de los ecosistemas y herramientas MLOps. En este aspecto, se diferenciarán diversas áreas cuyo estado conviene analizar de forma independiente, ya que en general el término MLOps abarca diferentes herramientas y metodologías. Estos áreas son:

1. Metodologías Software
2. DevOps
3. DevOps IA/MLOps

2.1. Historia y situación actual de las Metodologías Software

Para entender DevOps y MLOps, se debe partir desde el principio del desarrollo software. Es importante entender cómo se ha ido desarrollando Software a lo largo de los años, con diferentes metodologías, para entender las razones por las cuales las metodologías ágiles y el DevOps se han ido poco a poco implementando y ganando terreno frente a metodologías clásicas.

2.1.1. Primeros desarrollos software - Programación estructurada

Los primeros desarrollos de software datan de los años 40 y 50, con la primera generación de computadoras, ya que con ellas surgió la necesidad de desarrollar la serie de programas y

sistemas que requerían para funcionar. Esta época hizo uso de lo que se denomina *Structured Programming*, traducido como Programación Estructurada.

La Programación estructurada es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa mediante el uso extensivo de flujos de control estructurados de seleccion *(if/then/else)* y repeticiones *(while/for)*, estructuras de bloques y subrutinas. [7]

A finales de los años 50, con la aparición del lenguaje ALGOL [8], esta metodología de desarrollo se vio reforzada, ya que este lenguaje estaba dotado de estructuras de control consistentes y bien formadas. Además, en 1968, Edsger Dijkstra publicó un escrito titulado *"La Sentencia goto, considerada perjudicial"* [9], que abogaba por dejar de usar esta sentencia y tratar de hacer un código más legible y estructurado y evitar el denominado "código spaghetti".

Con el paso de los años, los desarrollos fueron ganando en complejidad al mismo tiempo que los ordenadores eran cada vez mejores y permitían desarrollar software más complejo, por lo que esta metodología como tal quedó obsoleta, pero sus principios se mantienen a día de hoy a la hora de hacer un código legible y correctamente estructurado y modular, e incluso sirven de base a la hora de desarrollar nuevos lenguajes de programación.

2.1.2. Metodología en Cascada

A finales de los años 60 y principios de los 70, los proyectos ya eran lo suficientemente complejos como para tener una lista de requisitos extensa, necesitar bastante documentación y una planificación a la hora de diseñar la solución. Para solventar este tipo de problemas, nace la metodología en Cascada.

La metodología en Cascada es una metodología de gestión de proyectos y de desarrollo de software rígida, que toma la idea de un proyecto bien definido desde el principio para llevarlo a cabo mediante una serie de pasos determinados. Su pilar fundamental es la correcta definición de los requisitos, que deberán ser inamovibles de principio a fin, para así planificar y estimar todo el trabajo antes de llevarlo a cabo. [10]

Cada paso del flujo de trabajo no puede comenzar a menos que el paso anterior haya sido completado, lo que puede llegar a ser un problema a la hora de afrontar proyectos con cambios habituales. Si el cliente quiere cambiar la dirección del proyecto cuando este está en proceso, surgen muchos problemas de alcance, presupuesto y tiempo de entrega debido a las dependencias entre fases.

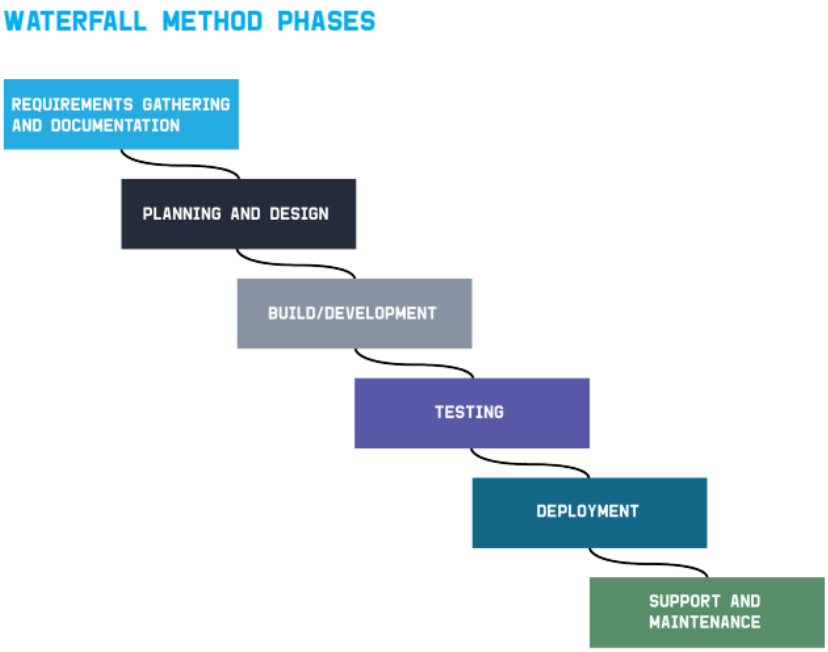

Figura 2.1: Fases de la metodología en Cascada

Generalmente, la metodología en cascada consta de las siguientes fases, como se puede observar también en la figura 2.1:

- Recolección de requisitos y documentación.
- Planificación y Diseño.
- Desarrollo.
- Pruebas.
- Despliegue de la solución, puesta en producción.
- Soporte y mantenimiento.

2.1.3. Metodologías Ágiles

Las metodologías ágiles nacen directamente en la industria del desarrollo de software, cuando las compañías del sector comprendieron que la forma tradicional de trabajo en cascada retrasaba mucho la entrega del producto final. Los procesos basados en contratos cerrados, requisitos inamovibles, y el hecho de que cada cambio suponía un esfuerzo enorme, resultaron ser muy ineficientes para los nuevos desarrollos que se querían abordar, y conducían a entregables de mala calidad.

En el año 2001, los CEOs de las principales empresas de software se reunieron en la ciudad estadounidense de Utah, y allí crearon el llamado "Manifiesto Agile". [11]

Este manifiesto tiene principalmente 4 valores clave:

- Los individuos y las interacciones sobre los procesos y las herramientas.
- Un producto funcional sobre una documentación exhaustiva.
- La colaboración con el cliente sobre la negociación de un contrato.
- Responder al cambio frente a seguir estrictamente el plan.

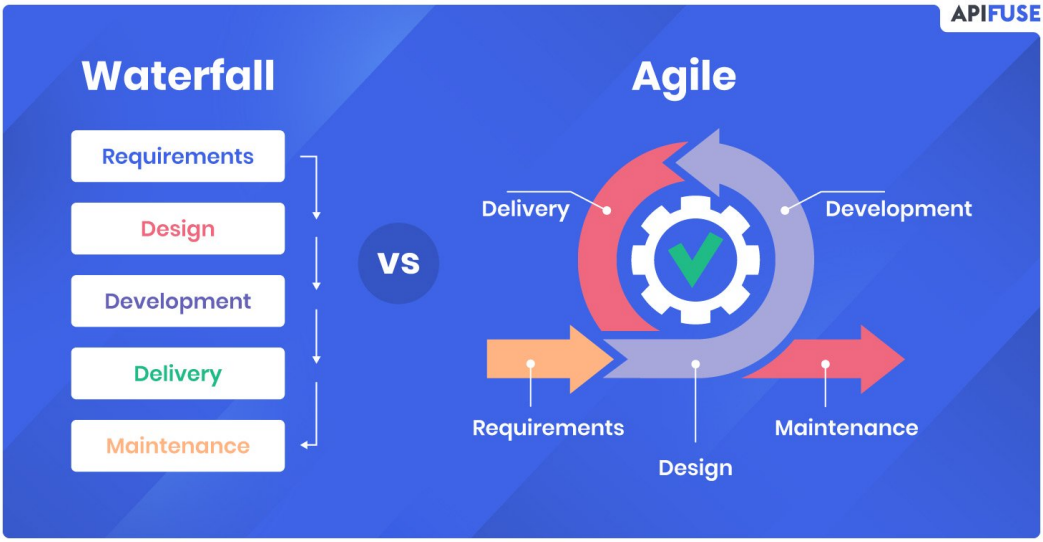

Figura 2.2: Metodología en cascada vs Metodología Agile

En la figura 2.4 se observa una pequeña comparativa entre las dos metodologías expuestas. En base a la figura y a todo lo expuesto anteriormente, se pone en evidencia que la metodología en cascada es claramente una metodología más cerrada y secuencial, mientras que las metodologías ágiles intentan ser iterativas y más abiertas a posibles cambios a lo largo del desarrollo.

Aunque ambas metodologías se siguen utilizando a día de hoy en una gran variedad de proyectos, la flexibilidad que ofrece *Agile* resulta clave para las empresas y los equipos de desarrollo que están siempre obligados a adaptarse a un entorno en permanente cambio.

Así, se identifican una serie de beneficios a la hora de afrontar proyectos software utilizando metodologías *Agile*:

1. **Simplificación de la sobrecarga de procesos.**

   Este punto va ligado directamente con el siguiente, la mejora de calidad. La principal ventaja de las metodologías ágiles es su carácter iterativo. Esta serie de iteraciones están planeadas para

que cada entregable cumpla una serie de requisitos y de normas, lo que en última instancia generará que el producto final cumpla todos los estándares planeados sin tener que llevar a cabo un esfuerzo final enorme por adaptarse a la normativa.

2. **Mejora de la calidad.**

El objetivo de cada entregable es proporcionar la mínima funcionalidad con la máxima calidad, lo cual no implica necesariamente que la funcionalidad sea muy pobre, sencillamente que será pequeña y se irá incrementando con cada iteración. Esta filosofía es tremendamente útil a la hora de verificar realmente los requisitos del cliente, ya que en muchas ocasiones lo que el cliente cree que necesita es muy diferente a lo que necesita en última instancia. Las metodologías ágiles permiten al cliente ver en cada fase del desarrollo lo que se está haciendo y proporcionar el *feedback* necesario al equipo de desarrollo para que el producto se adapte a sus necesidades.

3. **Mejora de la previsibilidad a través de una mejor gestión del riesgo.**

Mientras que las metodologías en cascada están muy poco preparadas para que surjan problemas y cambios a lo largo del desarrollo, las metodologías ágiles nacen precisamente para intentar minimizar los riesgos y dar la mejor respuesta posible a los cambios y problemas que surgen durante el ciclo de vida del proyecto.

De hecho, la filosofía ágil es generalmente la de dar prioridad a los riesgos, abordando primero aquellas tareas más cruciales y difusas, validándolas con el cliente, y mejorando en aquellos aspectos en los que no quede del todo satisfecho iterando en los diferentes entregables. Así, la validación del producto final es más rápida y, en general, los riesgos se conocen y se manejan de una manera óptima.

4. **Mejora de la productividad** Aunque no hay prácticamente ninguna referencia ni estudio científico que lo demuestre, los equipos ágiles parecen mostrar una mayor productividad que aquellos que utilizan métodos tradicionales. El hecho de tener que lanzar un entregable con funcionalidad cada poco tiempo aumenta la productividad del equipo, y aunque sobre el papel puede parecer estresante, lo cierto es que abordar problemas pequeños para llegar a resolver uno mayor suele ser una metodología muy usada en muchos ámbitos de la vida cotidiana y laboral. Eso sí, es importante tener en cuenta que el equipo se debe coordinar desde el principio, y que el tiempo para cometer errores es menor, lo que puede llevar a una carga algo mayor, especialmente en las fases iniciales de los proyectos.

Los diferentes equipos que participan en un proyecto siguiendo una metodología en cascada suelen tener poca interacción entre ellos, partiendo de desarrollos completamente segmentados por equipos y luego poniendo todo en conjunto en la fase de integración y pruebas. Son en estas

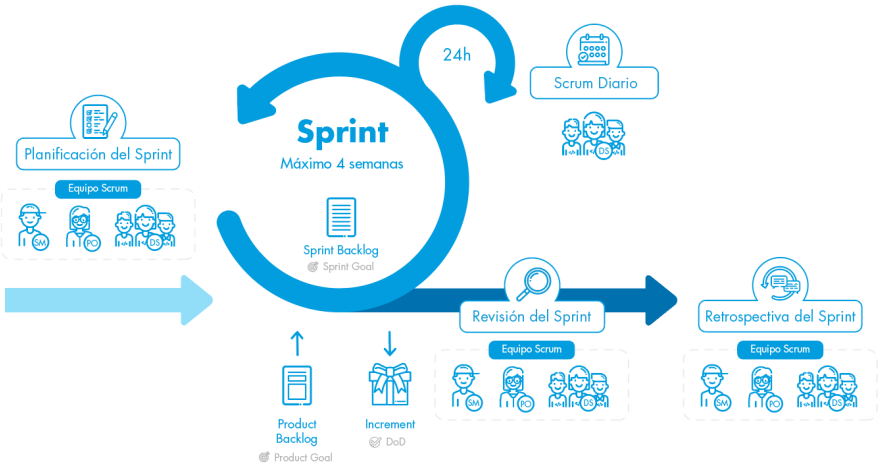

Figura 2.3: Modelo de la metodología *Scrum*, una metodología Ágil

dos últimas fases donde los equipos se coordinan con la maxima eficiencia, pero en el momento de llegar a estas etapas ya puede haber retrasos, fallos o problemas heredados de anteriores fases que sean complicados de abordar.

Las metodologías ágiles permiten tener un producto entregable mucho antes aunque con una funcionalidad menor, lo que provoca que los equipos se coordinen prácticamente desde el principio para que el mínimo producto viable nazca con buena calidad y en plazo.

5. ***Feedback* continuo del cliente**

Con las metodologías ágiles se obtiene una continua retroalimentación por parte del cliente, lo que a futuro mejora la calidad del producto y la satisfacción del propio cliente.

6. **Motivación del equipo**

En general, fijando objetivos a corto plazo y cumpliendo *quick wins*, es decir, cumpliendo pequeños objetivos para poder entregar producto de forma iterativa, el equipo se siente más motivado y la satisfacción de equipo y cliente será mayor.

Existen diversas metodologías ágiles, todas siguiendo los mismos pilares fundamentales pero con diferentes implementaciones de los mismos. Algunos ejemplos son:

- Kanban
- Scrum (figura 2.3)
- Lean
- Programación Extrema

Incluso en ocasiones las empresas y equipos de trabajo mezclan características de estas metodologías para generar sus propios modelos de trabajo, siempre basados en los pilares *agile*.

2.2. Surgimiento de las técnicas DevOps

Una vez explicado el contexto actual de las metodologías ágiles, es importante saber el por qué del nacimiento de DevOps.

En la mayoría de empresas, y a pesar del auge de la metodología ágil a la hora de afrontar los proyectos software, los equipos de desarrollo y los equipos de operaciones han estado completamente aislados durante años.

La metodología DevOps empezó a fraguarse entre el 2007 y el 2008, cuando los equipos de operaciones TI y los de desarrollo de software se dieron cuenta de que había un gran problema de alineamiento entre ellos [12]. Tradicionalmente, los equipos de desarrollo software existían independientemente de los equipos que desplegaban y ponían en producción ese código, y cada equipo tenía diferentes objetivos, llegando al punto en el que en ocasiones competían entre ellos. Además, los líderes de equipo eran distintos, los indicadores de rendimiento eran diferentes, y, desde el punto de vista físico, en la gran mayoría de ocasiones estaban en plantas o en edificios diferentes.

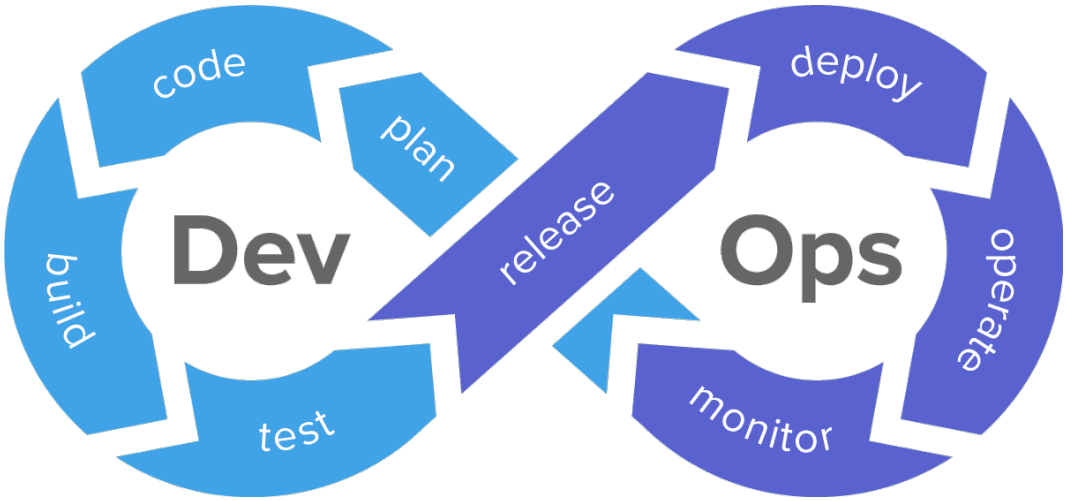

Figura 2.4: Esquema básico de la filosofía DevOps

DevOps comenzó en foros y encuentros locales entre gente perteneciente a estos dos equipos, y progresivamente las compañías fueron dándose cuenta de que no tenía sentido que los desarrolladores y los profesionales de operaciones o TI estuvieran tan separados entre ellos, y que esta separación en numerosas ocasiones traía diversos problemas asociados como largas jornadas de trabajo, publicaciones con una calidad nefasta, e insatisfacción por parte de los clientes.

Aunque se utilicen metodologías ágiles, en muchas ocasiones el despliegue y puesta en produc-

ción del código es una fase crítica y en ocasiones muy difícil de llevar a cabo, y precisamente para facilitar esas fases de despliegue e integración nace DevOps. Según una encuesta llevada a cabo por Atlassian, el 99 % de los equipos que están llevando a la práctica DevOps están seguros del éxito del código que estaban produciendo [13].

Además, en esa encuesta se encuentran otros datos interesantes, como que el 50 % de las organizaciones afirman haber practicado DevOps durante más de tres años.

DevOps afecta a todas las fases del ciclo de vida del desarrollo y las operaciones, ya que implica:

- Uso de las metodologías ágiles por definición.
- Integración y entrega continuas, piedra angular de las prácticas DevOps.
- Flujos de trabajo y repositorios Git de código para automatización y control de versiones.
- Gestión de servicios de TI y la gestión de incidentes y eventos no planificados.

Pese a que todas las empresas actualmente conocen DevOps y saben de los tremendos beneficios que puede aportar a sus compañías, la encuesta de Atlassian muestra que solamente el 50 % de las organizaciones afirman haber practicado DevOps durante más de tres años, lo que indica poca madurez en la gran mayoría de empresas en cuanto a implantación de esta metodología.

En el punto 2.3 se expone la historia del surgimiento del término MLOps, y de cómo las técnicas DevOps son aplicables e incluso pueden ser extendidas para aplicarse a los proyectos de Inteligencia Artificial, que, aunque son proyectos software, son muy distintos a los desarrollos de los que hemos hablado hasta ahora en este capítulo, y requieren conocer todo lo expuesto para poder aplicarlo posteriormente y particularizarlo a este tipo de proyectos.

2.2.1. Historia de las herramientas usadas en DevOps

Aunque en el capítulo 3 se expondrán más en profundidad las herramientas DevOps más utilizadas y se elegirán las necesarias para llevar a cabo la solución, en este capítulo me gustaría hablar de ellas desde un punto de vista más histórico.

Las técnicas DevOps no surgen porque salga una herramienta que permita hacer algo concreto, sino que es un cúmulo de circunstancias y de herramientas que se unen para poder llegar a la fase de madurez DevOps que se tiene actualmente. Así, a lo largo de los años, y para ayudar a los desarrolladores a tareas muy diferentes, fueron surgiendo diferentes herramientas que actualmente forman parte de cualquier ecosistema DevOps. Algunos ejemplos son:

1. **Git**

El desarrollo de Git empezó en abril del 2005, después de que varios desarrolladores del kernel de Linux perdieron el acceso a *BitKeeper*, el primer software de control de versiones completo, que se venía usando para mantener el proyecto desde el año 2002. El propietario del *copyright* de *BitKeeper*, Larry McVoy, retiró el uso gratuito del producto después de que surgieran otros softwares de control de versiones como SourcePuller o Mercurial, supuestamente y según McVoy, haciendo ingeniería inversa del suyo.

Linus Torvalds, creador y antiguo desarrollador principal del kernel de Linux, llegó a la conclusión de que ningún software de control de versiones gratuito cumplía los requisitos necesarios para sus necesidades, y se puso a trabajar en Git [14]. Tras comenzar el desarrollo el 3 de abril, el día 7 ya publicó la primera versión, siendo el día 18 de abril el primer día en el que se *mergearon* por primera vez multiples ramas de Git.

Actualmente, Git es el sistema de control de versiones más usado del mundo a nivel tanto profesional como educativo y personal, es la primera opción del 93.4 % de los desarrolladores según la última encuesta de StackOverflow, y por tanto, una herramienta fundamental para cualquier desarrollador [15].

El control de versiones del código es un pilar fundamental en DevOps, por lo que Git se hace una herramienta indispensable para llevar a cabo con éxito la aplicación de estas técnicas.

2. Docker

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software. La idea de lo que llamamos ahora "tecnología de contenedores"surgió por primera vez en el año 2000 como *FreeBSD jail*, una tecnología que permite la partición de un sistema *FreeBSD* en varios subsistemas o "jaulas". Las jaulas se desarrollaron con la idea de ser entornos seguros que un administrador de sistemas podía compartir con distintos usuarios, de forma que tuvieran un entorno definido de trabajo que no afectara a otros entornos dentro del mismo sistema. [16]

En 2001, se introdujo en las distribuciones Linux la implementación de un entorno aislado con el mismo objetivo, y a partir de ese momento se fueron combinando diferentes tecnologías, hasta que en el 2008 Docker apareció en escena.

La tecnología Docker incoporó una serie de conceptos y herramientas nuevas, como una interfaz de línea de comandos sencilla, un *daemon* de servidor, una extensa biblioteca de imágenes prediseñadas (DockerHub) y un servidor de registros. Todas estas tecnologías combinadas han permitido que las aplicaciones y soluciones software se puedan empaquetar y entregar en este tipo de contenedores, y que sean completamente independientes del sistema donde se quieran desplegar.

Esta capacidad de adaptación a cualquier sistema, y el hecho de que la imagen se genere siempre de la misma forma, permite a los desarrolladores generar productos que se pueden llevar muy rápidamente a producción, ya que con los contenedores se asegura que lo que funciona en un sitio funcionará en cualquier otro de la misma manera. Así, se evita la famosa frase "en mi sistema funcionaba bien".

Esta adaptabilidad y rapidez a la hora de llevar a producción los desarrollos de software es una piedra angular del DevOps y el MLOps.

3. **Jenkins**

Jenkins es un servidor de automatización de código abierto escrito en Java, que fue originalmente desarrollado con el numbre Hudson, desarrollo que comenzó en 2004 por la empresa *Sun Microsystems*. La primera versión de Jenkins data de febrero de 2005 [17].

Después de que Oracle reclamara el derecho al nombre y la marca registrada de Hudson, la comunidad aprobó en 2011 cambiar el nombre a Jenkins, y ambos proyectos siguieron de forma independiente.

Figura 2.5: Esquema básico de arquitectura DevOps. [1]

Actualmente, Jenkins es uno de los software DevOps más utilizados, si no el que más, ya que es un pilar fundamental en el ámbito de la Integración Continua (CI), ya que permite automatizar la integración de mejoras en el código de un proyecto una vez que han sido validadas, con el objetivo de detectar errores lo antes posible. Gracias a la integración continua, los desarrolladores pueden detectar y corregir errores constantemente, por lo que Jenkins es una herramienta indispensable a la hora de aplicar las metodologías *agile* y las técnicas DevOps.

Estas tres herramientas son en la actualidad piedra angular de las técnicas DevOps, y en muchísimas formaciones, cursos, seminarios y tutoriales de estas técnicas se pueden encontrar ejemplos básicos de arquitecturas en las que se utilizan Jenkins, Git y Docker para aplicar DevOps a nivel de ejemplo, como se puede ver en la figura 2.5.

2.3. MLOps: DevOps para Inteligencia Aritificial

Como se ha comentado en el capítulo 1, los proyectos de Inteligencia Artificial han crecido gradualmente y de forma acusada a lo largo de los últimos años, pero su puesta en producción siempre ha sido tremendamente compleja. Muchos de los algoritmos desarrollados nunca llegan a productivizarse, y en numerosas ocasiones se quedan como simples investigaciones o algoritmos que realmente no están aportando valor a las compañías y clientes.

Además, los proyectos que llegan a ponerse en producción lo hacen después de integraciones muy complejas y largas en el tiempo, lo que genera retrasos en la entrega del modelo final y del sistema de IA. Según un informe realizado por Algorithmia titulado *"2021 State of Enterprise Machine Learning"* [18], se pone en evidencia que muchas compañías aún no han conseguido cumplir sus metas en lo que a proyectos de Inteligencia Artificial se refiere.

Hay una diferencia fundamental entre desarrollar un modelo y ponerlo en producción. Según el informe de Algorithmia, el 64 % de las organizaciones tardan un mes o más en poner un modelo en producción, mientras que el 11 % de ellas lo hacen en una semana o menos. Este estudio se puede ver reflejado en la gráfica de la figura 2.6.

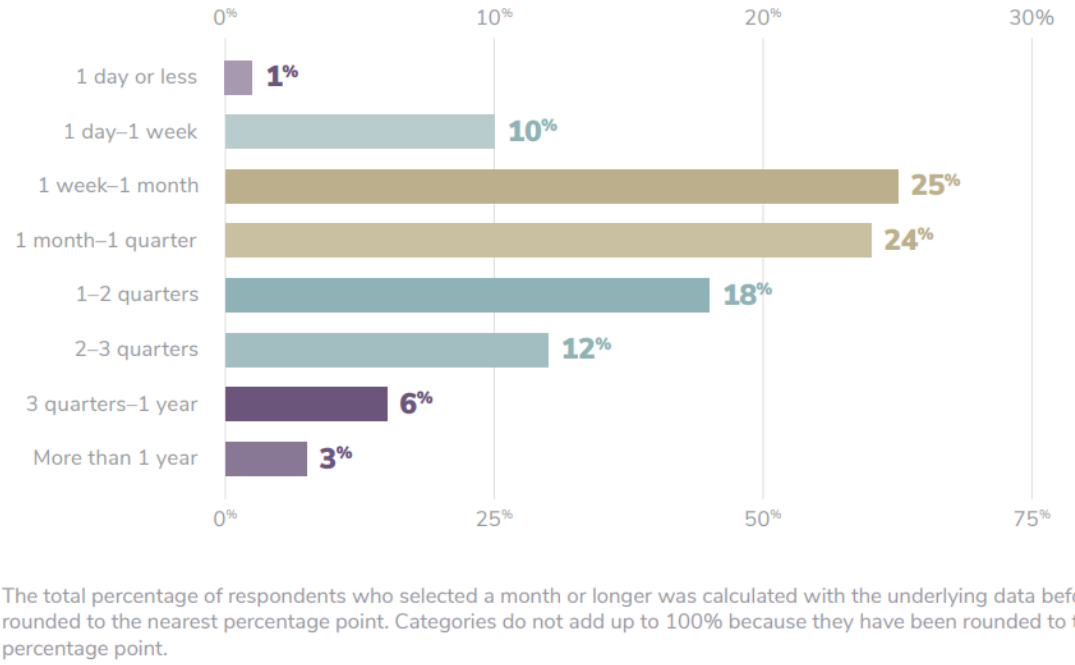

Figura 2.6: Encuesta sobre el tiempo de puesta en producción de algoritmos de IA

Realmente, este tiempo abarca tanto el desarrollo como la puesta en producción una vez desarrollado el modelo, pero en el informe también hay una pregunta específica acerca de la fase

de despliegue. Dicha pregunta pone en evidencia que el tiempo requerido para desplegar un modelo también se está incrementando, y que el 40 % de las empresas invierten entre 3 meses y hasta más de un año (3 %) en desplegar un modelo en producción, tal como se puede ver en la figura 2.7

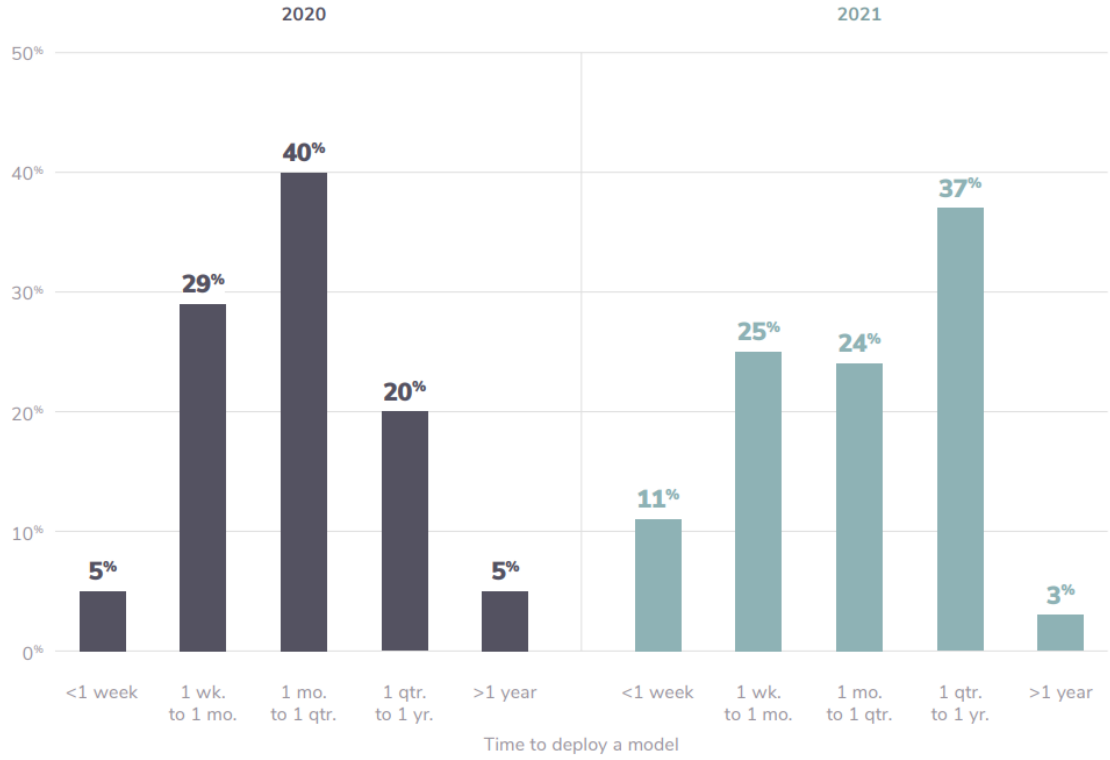

Figura 2.7: Encuesta sobre el tiempo de despliegue de algoritmos de IA

Con todos estos datos en la mano, queda claro que se debe buscar una solución que, tal como se ha ido haciendo históricamente, ponga en contacto a los desarrolladores de modelos y sistemas de IA con el área de operaciones de las compañías. Con ese objetivo, nace el término MLOps. Llegado a este punto, es posible que nos preguntemos «¿por qué acuñar un nuevo término cuando ya existe DevOps y, en el fondo, los desarrollos de IA son desarrollos software?».

La respuesta a esta pregunta es bastante sencilla, y es que los desarrollos de Inteligencia Artificial son bastante diferentes a los de desarrollos de software tradicionales. El pipeline de desarrollo de IA incluye tres niveles de cambio: Datos, Modelo y Código[2]. El término «nivel de cambio» se refiere a que en los sistemas de Inteligencia Artificial, el evento que provoca una nueva compilación y despliegue de la solución puede ser un cambio en uno o varios de estos elementos. Este principio se conoce como el principio de "Cambiar cualquier cosa cambia todo"[19], y se puede ver en la figura 2.8.

Adicionalmente, a lo largo de los años se han detectado diferentes problemas que presentan los

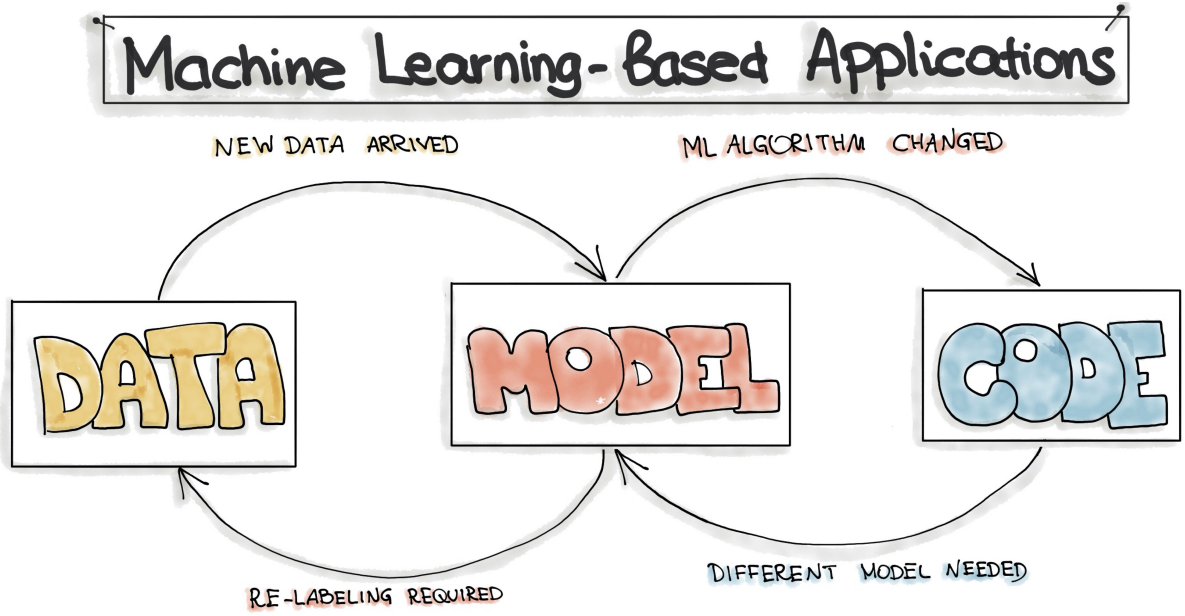

Figura 2.8: Los tres niveles de cambio, pilares de MLOps

algoritmos de Inteligencia Aritifical cuando están ya en producción, los cuales son:

- La calidad de los datos puede variar a lo largo del tiempo, o el modelo puede estar entrenado con datos que luego no se corresponden con la realidad de producción.
- El rendimiento de los modelos en producción puede degenerar a lo largo del tiempo por cambios en los datos de la vida real.
- Puede existir sesgo en función de la localización del modelo original. Un ejemplo claro de esto es entrenar un modelo de reconocimiento facial solamente con rostros de personas de un área determinada del mundo, y se suele conocer como **sesgo del modelo**.

Ahora que ya sabemos el entorno actual, conviene mirar hacia atrás y ver qué ha pasado para llegar a esta situación. Al principio de los 2000, cuando las empresas necesitaban implementar las primeras y más rudimentarias soluciones de Machine Learning, se usaba software propietarios como SAS, SPSS o FICO. Con el auge de las tecnologías open source y la disponibilidad cada vez mayor de set de datos, muchos desarrolladores comenzaron a usar Python y R para entrenar estos algoritmos. Sin embargo, la puesta en producción de los modelos siempre fue un problema, y según emergían las tecnologías de contenerización como Docker, se iban solucionando diferentes problemas como el entorno necesario o la escalabilidad del modelo (con Kubernetes) [20]. En la actualidad, la combinación de diferentes herramientas de IA y DevOps están dando como resultado el desarrollo y publicación de las conocidas como «Plataformas MLOps». Toda esta evolución temporal se puede ver en la figura 2.9

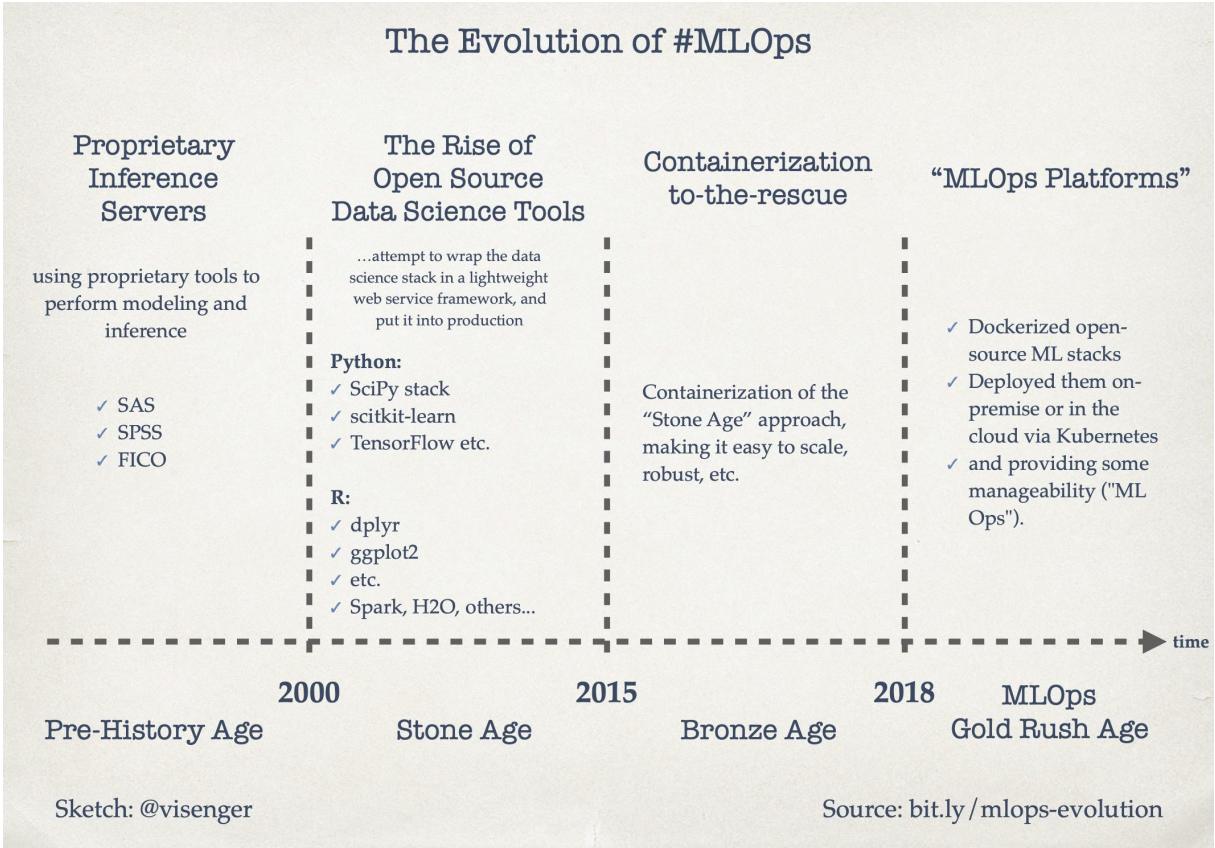

Figura 2.9: Evolución de MLOps [2]

Actualmente, y como ya se ha mencionado a lo largo de este capítulo, existen diferentes plataformas y frameworks de MLOps. Algunos ejemplos destacados son:

- Kubeflow.
- MLFlow.
- Valohai.
- Tensorflow Extend.
- Amazon SageMaker.
- H2O MLOps.

A lo largo de la fase de implementación, se realizará un estudio de algunas de las soluciones posibles y se hablará más en profundidad sobre varias de estas herramientas, concretamente este tema se trata en el capítulo 3, sección 3.4

Capítulo 3

Implementación de la solución

A lo largo de este capítulo se explicará el proceso de desarrollo e implementación que se ha llevado a cabo hasta llegar al sistema final. Antes de comenzar la implementación como tal, se ha llevado a cabo un proceso de documentación y análisis de las tecnologías disponibles y sus capacidades, que también queda reflejado en este capítulo, así como un análisis de las diferentes fases que componen un proyecto de datos con Inteligencia Artificial. Una vez definidos todos estos conceptos, se expone la solución desarrollada y los resultados obtenidos.

3.1. Ciclo de Vida de los Datos en proyectos de Inteligencia Artificial

Antes de comenzar la implementación, es importante entender qué fases componen un proyecto de Data Science con Inteligencia Artificial.

Generalmente, todos los proyectos de Inteligencia Aritificial, sean de Machine Learning o de Deep Learning, tienen fases comunes que van desde la comprensión del negocio hasta el despliegue del modelo, pasando por otras fases como la transformación de los datos o el análisis exploratorio. Además, este proceso no es lineal sino circular o iterativo, ya que una vez se despliega el modelo se debe monitorizar y revisar para poder seguir adaptándose al negocio.

Se suelen definir las siguientes fases en los proyectos de Inteligencia Artificial, representadas en la figura 3.1.

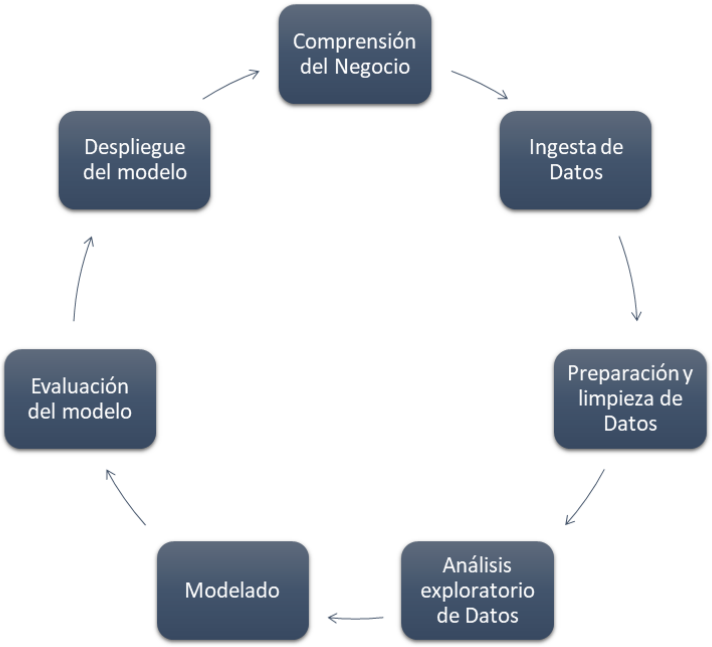

Figura 3.1: Fases tradicionales en un proyecto de Inteligencia Artificial

Este proceso se conoce como ***Cross Industry Standard Process for Data Mining*** o CRISP-DM, y se trata del modelo analítico más usado. En la figura 3.2 se puede ver una de las representaciones más utilizadas a la hora de definir el proceso. El objetivo de ambas figuras es el de ilustrar que el proceso es iterativo, y que en todo momento se debe tener claro el resultado y el objetivo de todas las fases para poder desarrollar el proyecto de la mejor manera posible.

1. **Comprensión del Negocio y los Datos**

El primer paso antes de comenzar el desarrollo es el de comprender el caso de uso, el sector en el que se va a trabajar, los diferentes problemas que se quieren abordar, así como todo lo referente a los datos que se necesitan para comenzar a trabajar (dónde están esos datos, en qué formato están y cómo se podrían explotar).

Comprender el negocio y los datos es clave para enfocar los objetivos del proyecto y obtener los datos necesarios para satisfacer esos objetivos. En esta fase, también es importante conocer el formato y la localización de los datos para poder extraerlos y explotarlos de forma eficiente.

2. **Ingesta de Datos**

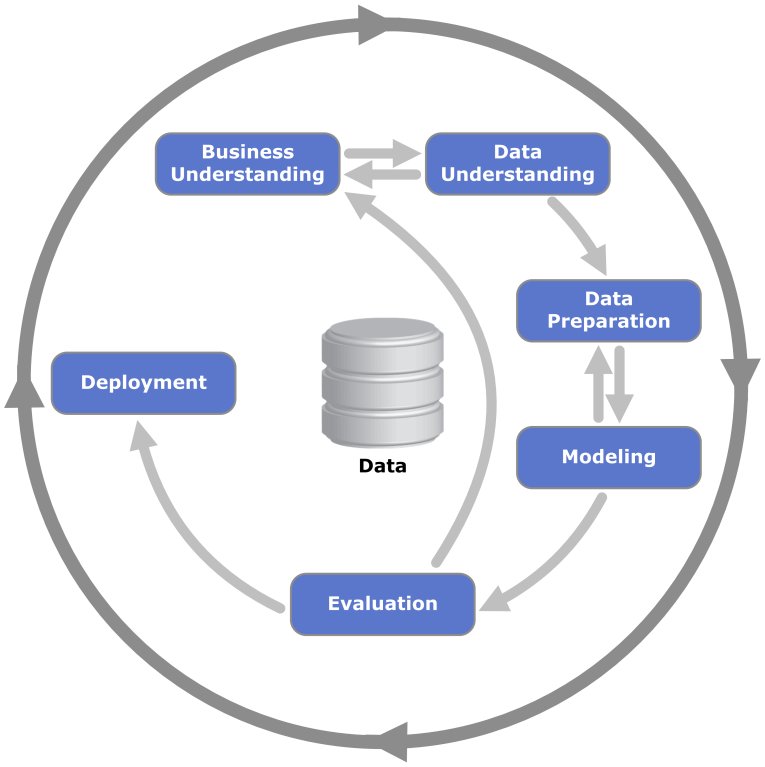

Figura 3.2: Modelo CRIPS-DM para proyectos de Minería de Datos

La ingesta de datos es la fase de obtención de los datos, que cambiará mucho en función de las fuentes y la frecuencia de los mismos.

Tanto en esta fase como en la anterior es vital hacer una buena evaluación de la volumetría de los datos y el tipo de arquitectura temporal (*batch* o *streaming*), de forma que el sistema diseñado y el método de ingesta seleccionado no colapsen a medida que el volumen va incrementándose. Este punto es vital, ya que muchos proyectos de Data Science presentan numerosos problemas a la hora de llevarlos a producción por no haber estimado bien el volumen, o sencillamente no funcionan bien porque obtienen muy pocos datos en relación a los que realmente se están generando.

3. Preparación y limpieza de Datos

Aunque esta fase va en parte ligada a la anterior, es de interés separarla porque suele ser una de las fases más complejas y, en la gran mayoría de proyectos, la fase más larga del proyecto. Para que los modelos de Inteligencia Artificial funcionen correctamente, se necesitan datos con la mayor calidad posible, por lo que la fase de preparación y limpieza resulta absolutamente vital.

Según un informe realizado por Anaconda en el año 2021 [21], y como se puede ver en la

figura 3.3, los Data Scientist invierten de media un 39 % del tiempo de un proyecto en preparar y limpiar los datos, porcentaje superior al que se invierte en las tres fases que involucran al modelo juntas.

Aunque en muchas ocasiones esta es una tarea que se intenta automatizar y estandarizar, la realidad es que es mejor tener una supervisión humana grande en esta fase, ya que las salidas de la misma y el asegurar la calidad de los datos es primordial para obtener mejores resultados en la fase de modelado.

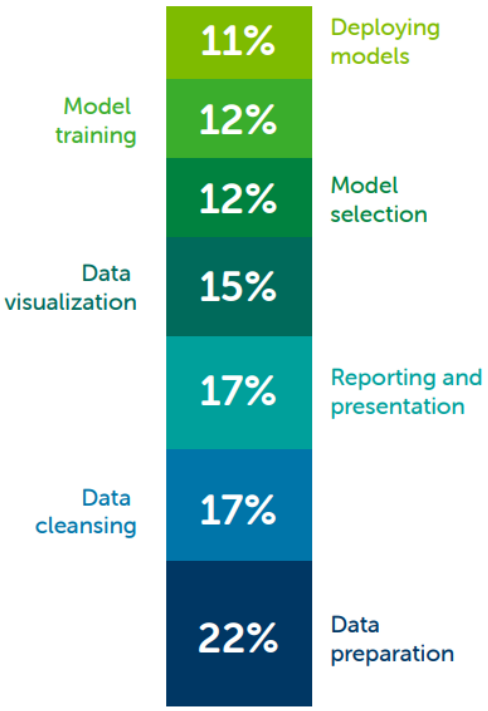

Figura 3.3: Porcentaje de tiempo invertido en cada tarea de Data Science

4. **Análisis Exploratorio de Datos**

El análisis exploratorio, también conocido como visualización de datos, es la fase en la que se interpretan y visualizan los datos limpios y preparados para poder empezar a tomar decisiones sobre ellos y poder interpretarlos de la mejor manera posible. Visualizar y plasmar el valor de los datos a través de herramientas de *reporting*, gráficos y tablas es una fase esencial de un proyecto de datos, y permite responder muchas cuestiones críticas para poder continuar con el desarrollo.

5. **Modelado y Evaluación del Modelo**

El modelado es considerado el corazón del análisis de datos. Un modelo toma los datos preparados como entrada y proporciona los datos deseados como salida. Este paso abarca

diferentes decisiones como elegir el modelo apropiado o seleccionar los hiper-parámetros que hagan el modelo mejor en rendimiento. Este rendimiento se mide por diferentes métricas, que dependiendo del modelo serán unas u otras, de forma que se evalúa el modelo para encontrar

6. **Despliegue del Modelo**

Una de las partes críticas de cualquier desarrollo y seguramente la más relacionada con DevOps es la de despliegue. Después de una evaluación rigurosa, el modelo se pone en el canal y el formato deseado, y aunque pareciera que este es el final del ciclo, realmente es otro comienzo. A partir del despliegue, se deben monitorizar los posibles problemas y cambios que van surgiendo a lo largo del tiempo, para seguir comprendiendo mejor el negocio y poder repetir el proceso iterativo de desarrollo.

Aunque en los pasos previos las técnicas de MLOps son importantes, es a partir de aquí donde el uso de estas técnicas mejoran los desarrollos de Inteligencia Artificial y aportan un tremendo valor, ya que con estas técnicas los Data Scientists son capaces de detectar de forma mucho más eficiente estos problemas, corregirlos y generar un sistema que sea capaz de desplegar nuevos modelos rápidamente, consiguiendo un sistema eficaz, con los mejores resultados posibles, y sostenible en el tiempo.

3.2. Grados de Madurez en MLOps

Antes de implementar el sistema, conviene entender cuáles son los diferentes grados de madurez posibles en una arquitectura MLOps. Existen diferentes modelos del grado de madurez de una arquitectura MLOPs, pero los dos más importantes ahora mismo son el modelo de Google [22] y el modelo de Microsoft [23].

Los modelos de madurez están diseñados para identificar brechas en el intento de implementar un entorno MLOps en una organización, y es una manera de mostrar cómo aumentar la capacidad de MLOPs en incrementos en lugar de abrumar a los arquitectos con los requisitos de un entorno totalmente maduro.

Para este TFM y por tanto para la arquitectura desplegada, se ha decidido seguir el modelo de grados de madurez de Microsoft, ya que es un modelo más detallado y concreto y que creo que representa mucho mejor la realidad de las diferentes implementaciones de MLOPs que se encuentran ahora mismo en la industria.

Este modelo de grados de madurez expone **5 niveles**, tal y como se observa en la figura 3.4, y en cada nivel se especifican **4 pilares fundamentales: personas, creación del modelo,**

lanzamiento del modelo e integración de aplicaciones.

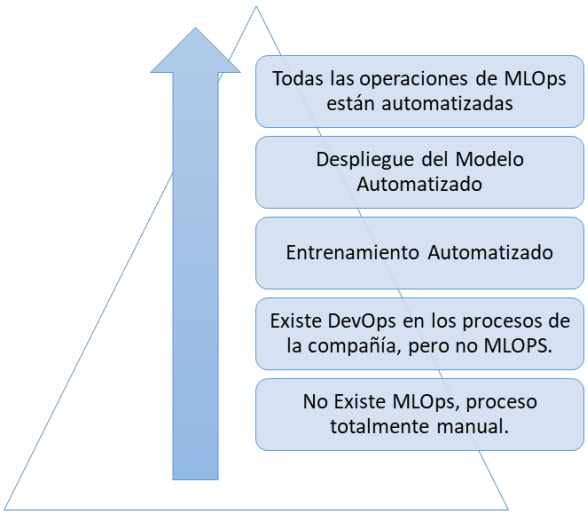

Figura 3.4: Grados de Madurez de MLOps en las compañías

3.2.1. Nivel 0: No existe MLOps

Personas

En lo referente a las personas que forman parte de los procesos MLOPs, se diferencian 3 perfiles fundamentales:

- **Científicos de Datos**: En silos aislados, no forman parte de las comunicaciones regulares con el equipo más grande.
- **Ingenieros de Datos**: En caso de existir, también en silos aislados.
- **Ingenieros de Software**: *Ídem*, en silos aislados, reciben el modelo de forma remota de los demás miembros del equipo.

Creación del modelo

- Los datos son recopilados **manualmente**.
- Es probable que el **proceso** no esté administrado.
- No se realiza un **seguimiento** previsible de los **experimentos**.
- El **resultado final** puede ser un único archivo de modelo que se entrega manualmente con entradas o salidas.

Lanzamiento del Modelo

- Proceso **manual**
- El script de puntuación puede crearse manualmente después de los experimentos, **sin control de versiones**.
- Versión controlada por el científico de datos o por el ingeniero de datos solamente.

Integración de Aplicaciones

- Depende en gran medida de la **experiencia del científico de datos** para realizar la implementación.
- Los lanzamientos son **manuales** cada vez.

3.2.2. Nivel 1: DevOps sin MLOps

Personas

No existe diferencia con el nivel 0 de madurez.

- **Científicos de Datos**: En silos aislados, no forman parte de las comunicaciones regulares con el equipo más grande.
- **Ingenieros de Datos**: En caso de existir, también en silos aislados.
- **Ingenieros de Software**: En silos aislados, reciben el modelo de forma remota de los demás miembros del equipo.

Creación del modelo

- Los datos son recopilados **automáticamente**.
- Proceso administrado o sin administrar.
- No se realiza un **seguimiento** previsible de los **experimentos**.
- El **resultado final** puede ser un único archivo de modelo que se entrega manualmente con entradas o salidas.

Lanzamiento del Modelo

- Proceso **manual**
- El script de puntuación puede crearse manualmente después de los experimentos, probablemente **con control de versiones**.
- Se entrega a ingenieros de Software.

Integración de Aplicaciones

- Existen pruebas de integración básicas para el modelo.
- Depende en gran medida de la **experiencia del científico de datos** para realizar la implementación.
- Los lanzamientos son **automatizados**.
- El código de la aplicación tiene pruebas unitarias.

3.2.3. Nivel 2: Entrenamiento automatizado

Personas

- **Científicos de Datos**: Trabajan directamente con ingenieros de datos para convertir el código de experimentación en scripts o trabajos repetibles.
- **Ingenieros de Datos**: Trabajan conjuntamente con los científicos de datos.
- **Ingenieros de Software**: En silos aislados, reciben el modelo de forma remota de los demás miembros del equipo.

Creación del modelo

- Los datos son recopilados **automáticamente**.
- Proceso **administrado**.
- Resultados del experimento con seguimiento.
- Tanto el código de entrenamiento como los modelos resultantes tienen control de versiones.

Lanzamiento del Modelo

- Proceso **manual**
- El script de puntuación tiene control de versiones con pruebas.
- Lanzamiento administrado por el equipo de ingeniería de software.

Integración de Aplicaciones

No hay diferencia con el nivel 1 de Madurez.

- Existen pruebas de integración básicas para el modelo.
- Depende en gran medida de la **experiencia del científico de datos** para realizar la implementación.
- Los lanzamientos son **automatizados**.
- El código de la aplicación tiene pruebas unitarias.

3.2.4. Nivel 3: Implementación del modelo automatizada

Personas

- **Científicos de Datos**: Trabajan directamente con ingenieros de datos para convertir el código de experimentación en scripts o trabajos repetibles.
- **Ingenieros de Datos**: Trabajan conjuntamente con los científicos de datos **y con los ingenieros de software** para administrar entradas o salidas.
- **Ingenieros de Software**: Trabajan conjuntamente con ingenieros de datos para automatizar la integración del modelo en código de aplicación.

Creación del modelo

Ninguna diferencia con el grado 2 de madurez.

- Los datos son recopilados **automáticamente**.
- Proceso **administrado**.
- Resultados del experimento con seguimiento.
- Tanto el código de entrenamiento como los modelos resultantes tienen control de versiones.

Lanzamiento del Modelo

- Proceso **automático**
- El script de puntuación tiene control de versiones con pruebas.
- Lanzamiento administrado por la canalización de entrega continua (CI/CD).

Integración de Aplicaciones

No hay diferencia con el nivel 1 de Madurez.

- Existen pruebas de integración para cada versión del modelo.
- Menos dependencia de la experiencia del científico de datos para implementar el modelo.
- El código de la aplicación tiene pruebas unitarias o de integración..

3.2.5. Nivel 4: Nuevo entrenamiento completo de MLOps automatizado

Personas

- **Científicos de Datos**: Trabajan directamente con ingenieros de datos para convertir el código de experimentación en scripts o trabajos repetibles. También trabajan conjuntamente con ingenieros de software para identificar marcadores para ingenieros de datos.
- **Ingenieros de Datos**: Trabajan conjuntamente con los científicos de datos **y con los ingenieros de software** para administrar entradas o salidas.
- **Ingenieros de Software**: Trabajan conjuntamente con ingenieros de datos para automatizar la integración del modelo en código de aplicación. También trabajan con científicos de datos para la implementación de la recopilación de métricas posterior a la implementación.

Creación del modelo

- Los datos son recopilados **automáticamente**.
- Proceso **administrado**.
- Resultados del experimento con seguimiento.
- Tanto el código de entrenamiento como los modelos resultantes tienen control de versiones.
- Los nuevos entrenamientos se desencadenan automáticamente en función de las métricas de producción.

Lanzamiento del Modelo

Ninguna diferencia con el grado 3 de madurez.

- Proceso **automático**
- El script de puntuación tiene control de versiones con pruebas.
- Lanzamiento administrado por la canalización de entrega continua (CI/CD).

Integración de Aplicaciones

- Existen pruebas de integración para cada **lanzamiento** del modelo.
- Menos dependencia de la experiencia del científico de datos para implementar el modelo.
- El código de la aplicación tiene pruebas unitarias o de integración.

3.3. Definición de la arquitectura general del proyecto

En este apartado se especifica la arquitectura genérica de la solución a implementar, teniendo en cuenta todo lo comentado anteriormente, en especial los grados de madurez de MLOps.

Así, se ha definido la arquitectura que se puede ver en la figura 3.5.

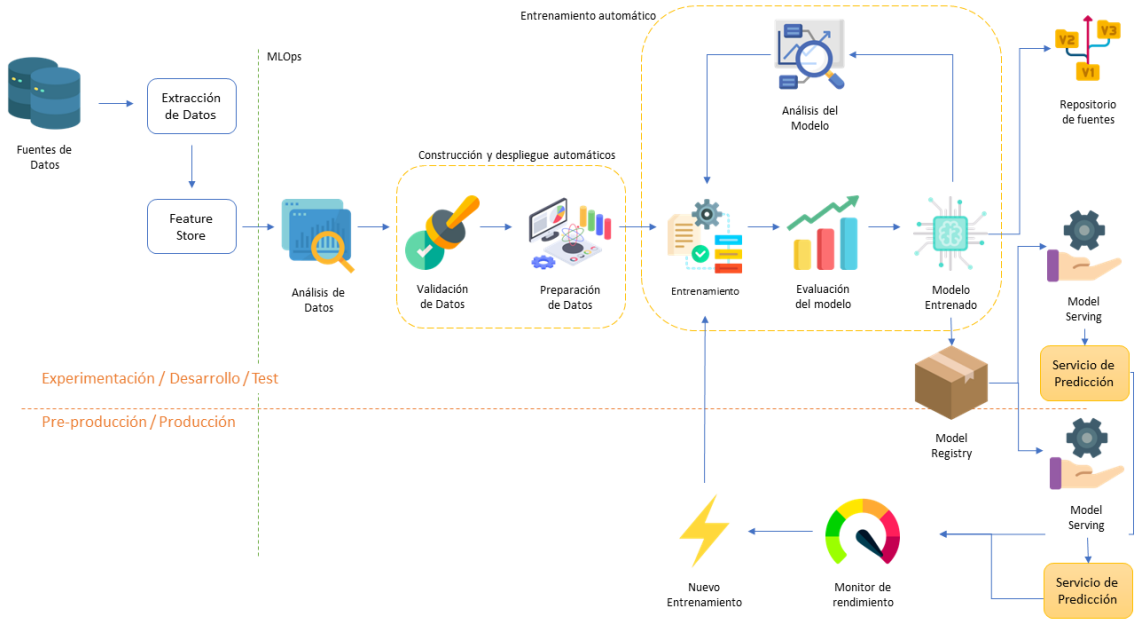

Figura 3.5: Arquitectura del Pipeline de la solución implementada.

Dicha arquitectura se puede dividir, como se puede ver en la imagen, en dos fases principales del desarrollo:

Experimentación / Desarrollo / Test

La primera parte de la fase de desarrollo en cualquier proyecto de datos sería la de conocer las fuentes de datos y extraerlos para poder posteriormente explotarlos. En esta fase, y tal y como se indica en la figura, ya encontramos un elemento específico de los pipeline MLOps, la conocida como Feature Store.

Una Feature Store, traducido como una «tienda de características», es básicamente un almacén de características, es decir, todos aquellas entradas para un modelo de Machine Learning o para un pipeline de un proyecto de datos. Aunque en el diagrama está puesto justo a la extracción de datos, es un componente transversal a todo el proceso de extracción, transformación y validación de los datos, por lo que en esta Feature Store se almacenarán tanto los datos extraídos de primeras, como los datos ya procesados y validados, para poder obtener una trazabilidad completa de cada dato utilizado para entrenar el modelo.

Las dos fases mecionadas de validación y preparación de los datos se realizarán de forma automática cada vez que se quiera reentrenar el modelo, por lo que la feature store se actualizará con nuevos datos en cada reentramiento, de forma que el modelo se retroalimente constantemente con características antiguas y nuevas.

Una vez que los datos están preparados, se procede al entrenamiento del modelo, que una vez más deberá ser un proceso automático cada cierto tiempo, siempre teniendo en cuenta el rendimiento actual del modelo en producción. Para ello, el modelo es entrenado, evaluado, y analizado para conseguir iterativamente el mejor modelo posible, que posteriormente será desplegado en producción.

Una vez terminado este proceso, tanto las fuentes como los modelos generados se almacenan en un repositorio de fuentes (para el código fuente) y un Model Registry para los diferentes modelos generados. Además, el modelo se desplegará en un servidor de pre-producción para probar su funcionamiento y su rendimiento, simulando en la medida de lo posible un entorno productivo.

Pre-Producción y Producción

Tanto en el entorno que simula producción como en el entorno final, el modelo se despliega y se monitoriza, de forma que se pueda ver claramente su rendimiento e intentar detectar lo antes posible caídas de rendimiento en las predicciones.

Si el monitor de rendimiento detecta que el modelo se ha degradado, se lanza la orden de entrenar un nuevo modelo, y el proceso mencionado anteriormente se vuelve a repetir para tratar de entrenar un modelo mejor que se ajuste a los cambios.

Así, el pipeline es completamente automático, y solamente requiere de la intervención del científico de datos a la hora de modificar el código, ya sea del procesamiento de los datos (por cambios de estructura en los mismos, o necesidad de añadir nuevas características para mejorar las predicciones), como de la fase de entrenamiento (por ejempplo, cambiando hiperparámetros para comparar diferentes modelos).

3.4. Análisis de Herramientas y Técnicas

En esta sección se analizan y comparan diferentes herramientas para el pipeline expuesto en el apartado anterior. En particular, se analizarán diferentes herramientas con la intención de responder las siguientes preguntas.

- ¿Cómo puedo llevar un registro del código fuente y de los datos (Feature Store)?.
- ¿Cómo puedo desplegar el modelo?.
- ¿Cómo monitorizo el rendimiento del modelo?.
- ¿Cómo coordino todo el pipeline?

3.4.1. Herramientas de control de versiones

Las herramientas de control de versiones permiten a los desarrolladores mantener una trazabilidad de los cambios realizados tanto en el código fuente como en los datos. Este apartado trata de comparar las diferentes herramientas que existen para realizar este seguimiento.

3.4.1.1. Control de versiones de código

3.4.1.1.1. Git

Página oficial: https://git-scm.com/

Como se mencionó en el apartado 2.2.1, Git es actualmente el sistema de control de versiones más usado del mundo a nivel tanto profesional como educativo y personal.

Git es gratuito, de código abierto, y multi plataforma, y permite llevar un seguimiento no lineal del código. Además, es capaz de gestionar tanto grandes proyectos como aquellos de tamaño reducido, y ofrece la posibilidad de ver tanto online como offline todo el árbol histórico del desarrollo.

En la tabla 3.1 se presentan los principales puntos positivos y negativos de Git.

3.4.1.1.2. Apache Subversion (SVN)

Página oficial: https://subversion.apache.org/

SVN es probablemente el sistema de gestión de repositorios más utilizado después de Git, y fue creado como alternativa a CVS para corregir muchos de los errores que este presenta. SVN sí presenta el concepto de operaciones atómicas, como se puede observar en la tabla 3.2

Puntos positivos	Puntos Negativos
▪ Herramienta multi plataforma y muy rápida ▪ Todo el árbol del histórico se puede consultar offline. ▪ Distribuida, herramienta *Peer to Peer*. ▪ Fácil de utilizar y consultar. ▪ La creación de ramas y su gestión es computacionalmente muy barata. ▪ Gestión por línea de comandos muy sencilla.	▪ Si el histórico es muy complejo, se vuelve complicado para un solo desarrollador ▪ Su compatibilidad con Windows va mejorando, pero no es perfecta comparado con Linux.

Cuadro 3.1: Puntos positivos y negativos de Git

Puntos positivos	Puntos Negativos
▪ Soporta operaciones atómicas que preveen la corrupción de archivos. ▪ Soporta directorios vacíos y las operaciones con ramas son más baratas que con CVS. ▪ Tiene mejor soporte en Windows que Git. ▪ Tiene plugins para casi todos los IDEs y herramientas Agile.	▪ No tiene suficientes comandos para la gestión del repositorio. ▪ Es mucho más lento que Git. ▪ Contiene bugs relacionados con la normalización de los nombres de los directorios y los archivos.

Cuadro 3.2: Puntos positivos y negativos de SVN

3.4.1.2. Control de versiones de datasets

3.4.1.2.1. Data Version Control (DVC))

Página oficial: https://dvc.org/

Data Version Control, conocido por su iniciales DVC, es una herramienta de gestión de experimentos Machine Learning y datasets que toma herramientas con las que los desarrolladores se sienten familiarizados (Git, CI/CD, etc).

Como se puede ver en la definición, DVC no solo guarda los datasets en un control de versiones, sino que también guarda todo modelo ML, con la característica de que además es una herramienta agnóstica de cualquier lenguaje, y se pueden guardar datasets y modelos en cualquier formato.

Estos datos pueden estar en local, aunque cuando los datasets van creciendo de tamaño sería convenientes guardarlos en una nube como puede ser Azure o Amazon AWS. DVC además cuenta con una interfaz de comandos igual a Git, por lo que es muy fácil de utilizar para aquellos desarrolladores familiarizados con este sistema de control de versiones.

Un esquema básico de utilización sería el presentado en la figura 3.6

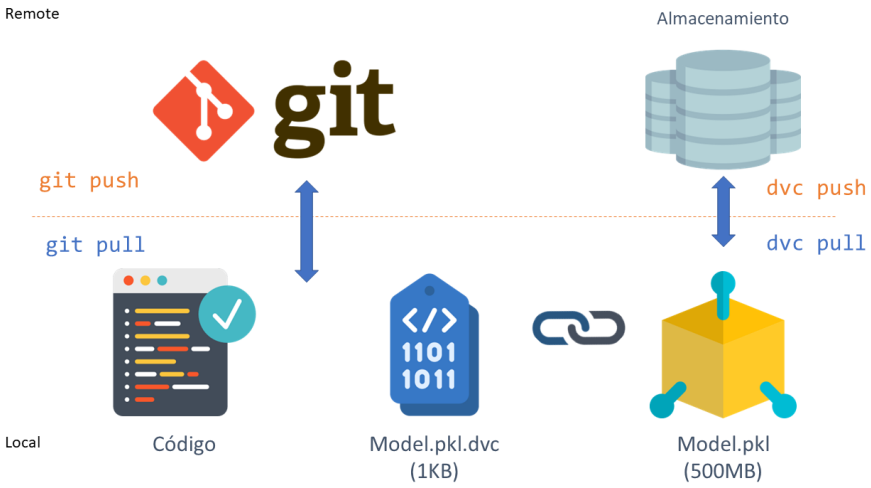

Figura 3.6: Flujo de trabajo con DVC

3.4.1.2.2. Git Large File Storage (Git LFS)

Página Oficial: https://git-lfs.github.com/

Git LFS es una aplicación que permite a los desarrolladores guardar ficheros binarios de gran tamaño junto a un repositorio git.

Aunque es una herramienta potente para mantener la trazabilidad de grandes archivos, esa es su

única utilidad, por lo que a la hora de realizar una trzabilidad de experimentos de Inteligencia Artificial resulta bastante ineficiente. Mientras que Git-LFS solo es capaz de gestionar las versiones de los datos y el código, DVC también hace un tracking del código.

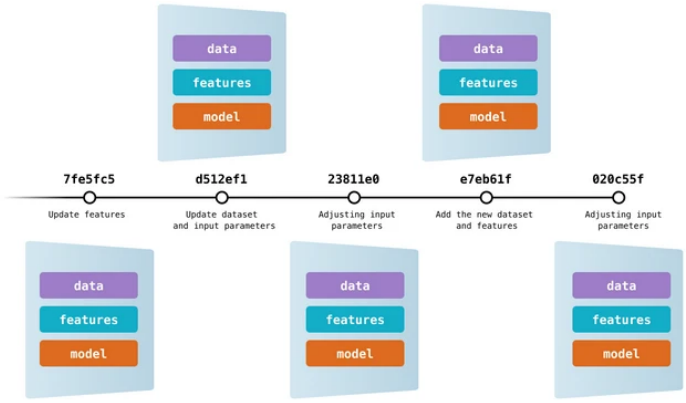

Figura 3.7: Tracking con DVC.

Esta diferencia de diseño de ambas herramientas se puede ver en las figuras 3.7 y 3.8, obtenidas de la documentación de DVC - https://dvc.org/

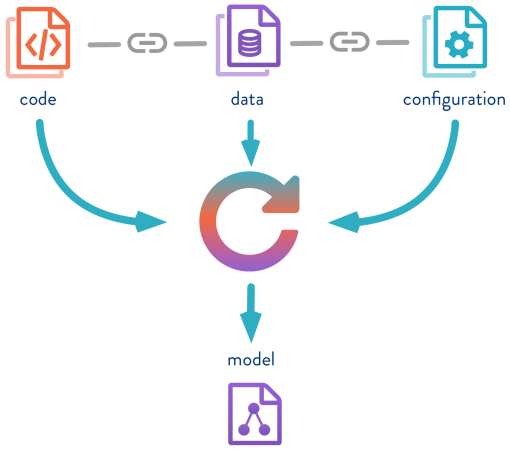

Figura 3.8: Diseño DVC para la reproducibilidad de experimentos.

3.4.2. Optimización de hiperparámetros

El «tuneado» u optimización de hiperparámetros es el problema mediante el cual se selecciona un conjunto de hiperparámetros óptimo para un algoritmo. Un hiperparámetro es un parámetro

cuyo valor es utilizado para controlar el proceso de aprendizaje de un algoritmo de inteligencia artificial.

En un proyecto de Inteligencia Artificial, un modelo de machine learning o deep learning puede requerir de diferentes constantes, pesos o tasas de aprendizaje para generalizar diferentes patrones de datos. Estas medidas se denominan hiperparámetros, y deben ser optimizados para que el modelo pueda resolver de la mejor manera el problema.

Este proceso de optimización tiene como objetivo encontrar un conjunto de hiperparámetros que minimicen la función de pérdida, la cual debe estar predefinida, en datos independientes. Esta generalización de rendimiento habitualmente es estimada mediante *Cross Validation*, métrica que utiliza diferentes porciones de los datos para estimar mediante varias iteraciones si el modelo funcionará en la práctica.

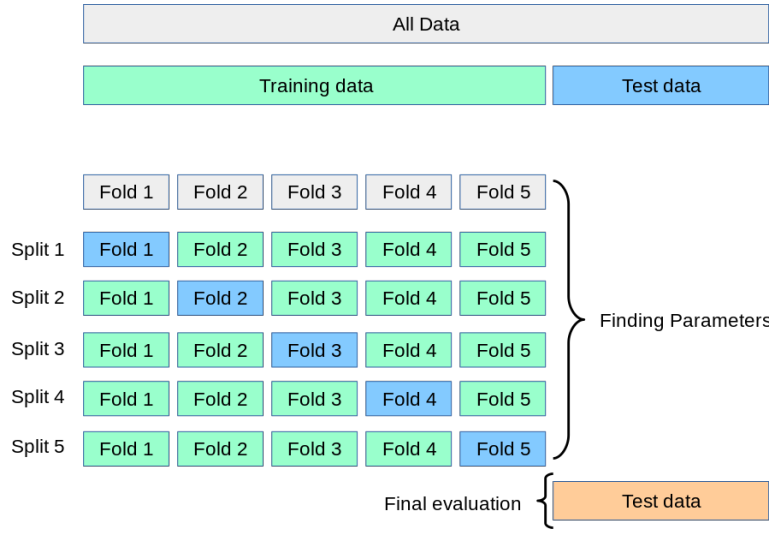

Figura 3.9: *Cross Validation* para evaluar el estimador de rendimiento del modelo

Existen diversas técnicas y herramientas que permiten a los desarrolladores optimizar los hiperparámetros, y aunque muchas ya de esas técnicas ya vienen incluidas en librerías de modelado como scikit-learn (caso de Grid Search o Random Search), existen frameworks más avanzados para realizar esta tarea crítica a la hora de desarrollar un modelo. Las dos herramientas más conocidas y utilizadas son Optuna y Ray Tune.

3.4.2.1. Optuna

Página oficial: https://optuna.org/

Optuna es un framework de optimización automática de hiperparámetros, diseñado específica-

mente para Machine Learning. Sus principales puntos fuertes son:

- Paralelización fácilmente implementable.
- Rápida visualización.
- Algoritmos de optimización muy eficientes.
- Ligero, versátil y completamente agnóstico de la plataforma.

Optuna se puede instalar como cualquier otra librería de Python con pip, y es una herramienta básica para encontrar los valores óptimos a la hora de generar los modelos.

3.4.2.2. Ray-Tune

Página Oficial: https://docs.ray.io/en/latest/tune/index.html

Tune es una librería de Python para ejecución de experimentos y tuning de hiperparámetros a cualquier escala. Soporta modelos de PyTorch, XGBoos, TensorFlow y Keras, y utiliza todos los algoritmos que están presentes en el estado del arte de la optimización de hiperparámetros, tales como *Population Based Training*, *BayesOptSearch* o *HyperBand/ASHA*. Además, Tune se integra con otras herramientas como la misma Optuna, expuesta en el apartado anterior.

Tune tiene las siguientes ventajas respecto a Optuna:

- Mayor soporte de librerías.
- Ejecuciones multi-nodo en menos de diez líneas de código, con código muy sencillo.
- Sus algoritmos de búsqueda son realmente *wrappers* de otras librerías como HyperOpt, SigOpt, Dragonfly o Facebook Ax.
- Sus resultados se pueden ver automáticamente con TensorBoard.

Además de las ventajas expuestas, la documentación de Ray Tune es muy completa, lo que lo convierte en seguramente el mejor framework de optimización de hiperparámetros existente actualmente.

3.4.3. Herramientas y APIs para Despliegue de Modelos

A la hora de poner en producción los modelos, se necesita una interfaz de comunicación para que dichos modelos reciban nuevos datos y apliquen inferencia sobre ellos. Esta interfaz es

básicamente un API (Interfaz de Programación de Aplicaciones), un intermediario entre dos aplicaciones independientes que se deben comunicar entre ellas.

Además, los modelos presentan dependencias a la hora de utilizar ciertas librerías para funcionar, por lo que se deben desplegar en un entorno. Para que la gestión de dependencias sea correcta y el entorno sea homogéneo, lo ideal es utilizar un software de contenerización. En este apartado se tratan todas estas herramientas.

3.4.3.1. Flask y FastApi

Página Oficial de Flask: https://flask.palletsprojects.com/en/2.1.x/ Página Oficial de FastAPI: https://fastapi.tiangolo.com/

Flask y FastApi son las dos librerías más conocidas para la generación de APIs en Python. En la tabla 3.3 se observa una comparativa de ambas herramientas.

Flask	FastApi
▪ Librería muy minimalista, solo los componentes principales van empaquetados. ▪ Código sencillo. ▪ No permite asincronía. ▪ No genera documentación automáticamente. ▪ No tiene validación de datos.	▪ Más rápida que Flask. ▪ Separa el código del servidor del código de negocio. ▪ Tiene un sistema de validación de datos. ▪ Está construida sobre ASGI (Asynchronus Server Gateway Interface), es decir, soporta asincronía. ▪ Puede detectar un tipo inválido de datos sobre la marcha, retornando el valor que ha dado error. ▪ Genera la documentación sobre la marcha en Swagger u OpenAI.

Cuadro 3.3: Flask vs FastAPI

Actualmente, FastAPI es bastante más utilizada entre los data scientist por todas las ventajas y evoluciones que presenta con respecto a Flask.

3.4.3.2. Contenedores: Docker y Kubernetes

Página Oficial de Docker: https://www.docker.com/ Página Oficial de Kubernetes: https://kubernetes.io/

Los contenedores son una forma de virtualización del sistema operativo, y son una forma optimizada de crear, probar, poner en marcha y volver a poner en marcha aplicaciones en diversos entornos, ya sea en desarrollo, pre-producción o en el sistema final. Sus principales ventajas son la total portabilidad de la solución, el ahorro de recursos, la rápida puesta en producción tras el desarrollo y la mayor eficiencia a la hora de escalar.

Aunque existen diversos frameworks de contenerización, la gran mayoría de compañías y desarrolladores utilizan los contenedores Docker, ya que es el framework más conocido y dispone de una serie de herramientas que permiten a los desarrolladores probar, levantar y empaquetar sus soluciones de forma rápida.

Su aplicación en el ámbito de MLOps es básica, evitar que las dependencias de los programas que generan y utilizan los modelos de Machine Learning hagan que la solución no funcione en producción. Cuando el data scientist termina el modelo, todo el código que está en el repositorio se empaqueta para posteriormente ser desplegado en un contenedor. De esta forma, teniendo el contenedor en funcionamiento, se puede asegurar que todo va a funcionar de la misma manera y sobre el mismo entorno en cualquier parte donde despleguemos la solución.

A la hora de orquestar los contenedores Docker, el propio Docker tiene una herramienta llamada Docker Swarm, pero la más conocida y extendida es Kubernetes.

Kubernetes es un framework de orquestación de contenedores que permite gestionar, conectar, escalar y prácticamente realizar cualquier operación con contenedores Docker. Es decir, una vez que los desarrolladores han terminado su contenedor y desean desplegarlo en producción, Kubernetes permite disponer simultáneamente de todos los contenedores que sean necesarios, gestionar las redes y conexiones entre ellos y establecer reglas de carga para escalarlos cuando sea necesario.

En la figura 3.10 se puede observar un pipeline básico de desarrollo con Docker y Kubernetes.

3.4.4. Herramientas de control del pipeline

Las herramientas de control del pipeline permiten sincronizar las diferentes herramientas y ejecutar los procesos de forma secuencial o paralelamente. Aunque son dos herramientas diferentes

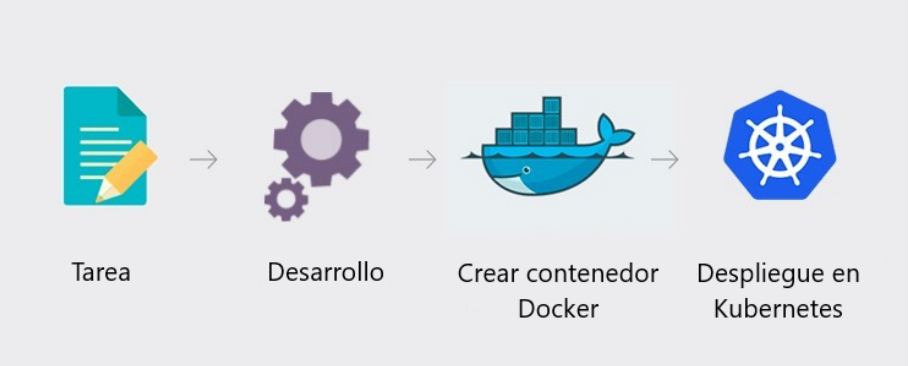

Figura 3.10: Pipeline básico de desarrollo con Docker y Kubernetes

para cometidos distintos, en este apartado se analizan dos de las herramientas más utilizadas para control de procesos y automatización: Apache Airflow y Jenkins.

3.4.4.1. Apache Airflow

Página Oficial: https://airflow.apache.org/

Apache Airflow es una herramienta de código abierto que permite la automatización del workflow. Es una herramienta escrita en Python que simplifica las tareas en lo que a desplegar y mantener un pipeline de datos se refiere.

Airflow tiene como componente principal los denominados DAGs (*Directed Acyclic Graphs)*, que permiten a los desarrolladores mantener, estructurar y organizar el siempre complejo proceso de extraer datos de entrada, transformarlos y cargarlos en una base de datos.

Estas tareas también se pueden programar en el tiempo y automatizar, y además Airflow cuenta con una interfaz web que la hace fácil de utilizar y que reduce enormemente la curva de aprendizaje de la herramienta.

Airflow además es compatible con Kubernetes gracias a su Kubernetes Executor, lo que permite ejecutar tareas en paralelo. Esto puede llegar a ser tremendamente útil para pipelines de Inteligencia Artificial, ya que si por ejemplo quisieramos comparar el resultado de dos modelos diferentes a la hora de predecir, podríamos desplegar dos caminos diferentes con ambos modelos para posteriormente comparar, y que cada uno de estos caminos se ejecutaran de forma independiente en pods de Kubernetes.

Una ejemplo de DAG se puede observar en la figura 3.11

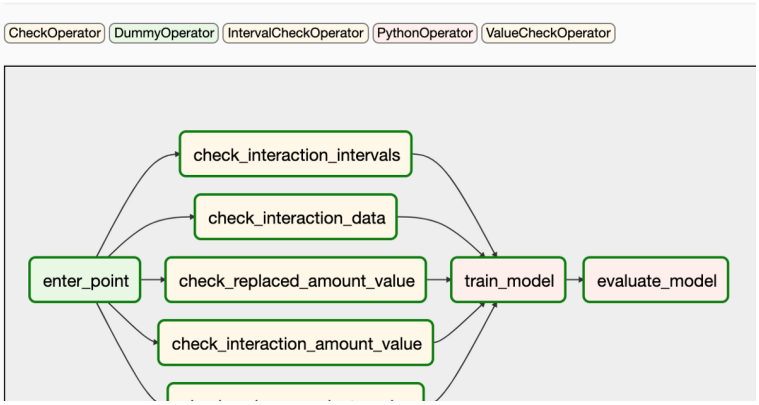

Figura 3.11: Ejemplo de DAG en Airflow

3.4.4.2. Jenkins

Página Oficial: https://www.jenkins.io/

Jenkins es un servidor de código abierto escrito en Java que permite a los desarrolladores y arquitectos DevOps ejecutar una secuencia de tareas de Integración Continua (CI) de forma automática. Es decir, permite automatizar tareas como la compilación, empaquetado, testing y despliegue de proyectos.

Jenkins es tremendamente popular ya que automatiza muchísimas tareas de desarrollo de software y permite detectar errores en fases de desarrollo tempranas. Además, tiene una comunidad enorme que ofrece soluciones rápidas a los problemas que puedan surgir con la aplicación, y un ecosistema de plugins compatible con infinidad de herramientas.

Un ejemplo de pipeline con Jenkins sería el siguiente:

1. El Data Scientist genera el código y lo sube a un repositorio Git.
2. Jenkins detecta la subida.
3. Se inicia el proceso de construcción del paquete y el contenedor, para posteriormente desplegar el contenedor.
4. Se ejecuta el contenedor con la solución, se entrena el modelo y se testea.
5. Si pasa todas las pruebas, se lanza a producción.

Toda esta orquestación se realizaría a través de Jenkins y el archivo Jenkinsfile, que es en donde los desarrolladores definen los pasos a seguir una vez el código se ha subido al repositorio. En la figura 3.24 se observa un ejempplo de pipeline con Jenkins.

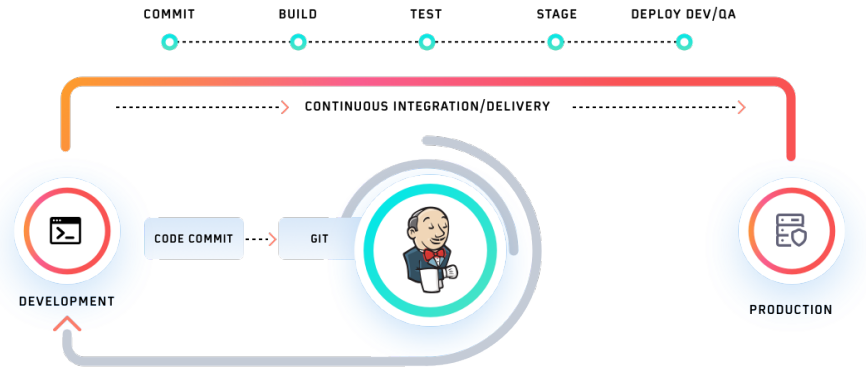

Figura 3.12: Ejemplo de pipeline con Jenkins

3.4.5. Herramientas específicas de MLOps

En este apartado se exponen algunas de las herramientas y frameworks específicos de MLOps más conocidos y utilizados en la actualidad. Aunque la finalidad de este trabajo es exponer herramientas gratuitas y de código abierto, en este apartado se mencionan algunas herramientas de pago por su amplia extensión en el mercado.

3.4.5.1. MLFlow

Página Oficial: https://mlflow.org/

Mlflow es una herramienta (librería de Python) desarrollada originalmente por Databricks destinada a gestionar el ciclo de vida de los modelos de Machine Learning. Permite registrar tanto los parámetros de los modelos como las métricas de evaluación a lo largo de los entrenamientos, así como tags para incorporar información adicional. De este modo permite la posibilidad de revisar y comparar los distintos entrenamientos y/o modelos con el fin de escoger y recuperar el deseado en momentos posteriores.

Además, es posible guardar junto a cada entrenamiento todo tipo de artefactos (archivos) que el usuario considere útiles, como por ejemplo el propio modelo, sus pesos, checkpoints, o incluso un archivo de configuración, entre otros.

Cada entrenamiento realizado se denomina "run", y estos a su vez se agrupan formando parte de un "experimento". A modo de ejemplo, un experimento podría corresponderse con el desarrollo de un árbol de decisiones, dentro del cual se guardarían múltiples runs con sus respectivas métricas, parámetros, tags y artefactos asociados.

Mlflow posee una interfaz de usuario, sencilla e intuitiva, en la que poder visualizar los distintos experimentos. Esta interfaz permite, entre otras funciones:

- Aplicar filtros a los experimentos y ejecuciones en función de cualquier métrica, parámetro o tag, con el fin de seleccionar entrenamientos que cumplan ciertos requisitos.
- Visualizar gráficamente la evolución de las métricas recogidas a lo largo de los entrenamientos, en función de las épocas o del tiempo de ejecución.
- Seleccionar y comparar determinadas ejecuciones.

Figura 3.13: Logo de MLFlow

Pese a todas estas ventajas, MLFlow también presenta una serie de puntos de mejora:

- Únicamente se centra en el seguimiento y versionado de modelos, pero la ejecución del pipeline queda fuera de su alcance y orquestación.
- No hace seguimiento de los datasets utilizados, a no ser que se guarden como artifact, pero no diferencia versiones.
- No posee control de roles, usuarios o forma de ser centralizado para una compañía.

Estas limitaciones pueden solventarse con herramientas de control del pipeline como Airflow, de orquestación como Jenkins o herramientas de control de versiones de datasets como DVC.

3.4.5.2. Kubeflow

Página Oficial: https://www.kubeflow.org/

Kubeflow es una plataforma basada en Kubernetes, diseñada para facilitar producción y mantenimiento de modelos de Machine Learning. La plataforma abarca el ciclo de vida completo de cada aplicación, desde la creación y entrenamiento de modelos hasta su puesta en producción y posterior mantenimiento.

La arquitectura basada en Kubernetes hace que Kubeflow tenga capacidad de orquestación, es decir, es capaz de crear pipelines combinando elementos contenerizados completamente independientes unos de otros, los cuales se corresponden con distintas fases dentro de un experimento de ML, como por ejemplo la carga de datos, pre-procesado, entrenamiento, evaluación, inferencia, etc.

Esta orquestación de contenedores le proporciona ciertas ventajas:

- Paralelización directamente integrada en la plataforma.
- Escalabilidad por diseño.
- Portabilidad, cada contenedor posee su propio entorno, lo que elimina los conflictos e incompatibilidades entre las versiones de los distintos contenedores o con el propio entorno donde se lleve a cabo la puesta en producción
- Reutilización, una vez creados los contenedores, estos pueden reutilizarse y combinarse de múltiples formas para la creación de nuevos pipelines.

Figura 3.14: Logo de Kubeflow

Por otro lado, Kubeflow también permite realizar un seguimiento de los experimentos que se llevan a cabo con cada uno de los pipelines. No obstante, aunque recoge ciertas métricas para evaluar la performance de cada modelo y/o experimento, está más enfocado a la reproducibilidad y evaluación del pipeline completo que al estudio en detalle de cada modelo. Por ello, es recomendable complementarlo con otras herramientas, como MLFlow, que puedan recopilar más información de los modelos con el fin de realizar comparaciones más detalladas de estos.

Uno de los puntos fuertes de esta plataforma es también una de las principales barreras de entrada, y es el uso de Kubernetes. Aunque no sea del todo necesario, para sacarle el máximo partido a Kubeflow es recomendable tener ciertos conocimientos de Kubernetes.

No obstante, Kubeflow posee un componente llamado Kale que permite la creación automática de pipelines a partir de código escrito en notebooks, facilitando en gran medida este proceso. Aunque para su correcto funcionamiento es necesario instalar también ROK, un componente de

pago desarrollado por Arrikto. Existe la posibilidad de probar Kubeflow+Kale+ROK de forma gratuita levantando una máquina virtual de MiniKF.

Otro de los mayores inconvenientes de Kubeflow es la dificultad de instalación, ya que sus requisitos son muy concretos, necesita una capacidad de computación alta y en numerosas ocasiones, la instalación falla sin dar al usuario logs útiles o que den muchas pistas para solucionar el problema.

3.4.5.3. H2O MLOps

Página Oficial: https://h2o.ai/resources/product-brief/h2o-mlops/

H2O es una plataforma de código abierto para machine learning distribuido en memoria, con escalabilidad lineal, que soporta la gran mayoría de modelos estadísticos y de machine learning. Dentro de esta plataforma se han desarrollado varios productos como H2O AutoML, una herramienta que se utiliza para la automatización de la selección del algoritmo, extracción de características, tuneado de hiperparámetros e iteraciones.

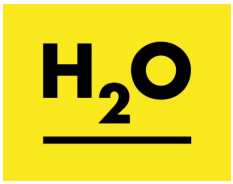

Figura 3.15: Logo de H2O

Dentro de las diferentes soluciones, existe también H2O MLOps, un entorno colaborativo de pago para Data Scientists y equipo IT para gestionar, desplegar, gobernar y monitorizar modelos de machine learning. Sus características principales son:

- Gestión de modelos, con trazabilidad completa de experimentos.
- Despliegue de los de modelos, poniéndolos a disposición de los usuarios como endpoints REST.
- Gobierno de modelos, guardando datos, artefactos, modelos, despliegues y el linaje entre unos componentes y otros.
- Monitorización del modelo una vez en producción, vigilando la degradación y el sesgo que pueda tener el modelo.

3.4.5.4. Valohai

Página Oficial: https://valohai.com/

Valohai es una plataforma de gestión de Inteligencia Artificial que ayuda a las empresas a automatizar la infraestructura de Data Science. La plataforma permite a los científicos de datos gestionar la orquestación de máquinas y contenedores, el control de versiones y los canales de datos.

Figura 3.16: Logo de Valohai

Valohai ofrece diversas funciones:

- Barridos de hiperparámetros en paralelo.
- Scripts personalizados.
- Visualización de sesiones de entrenamiento.
- Exploración de datos.
- Extensión para Notebooks Jupyter.
- Supervisión de los resultados en producción e implementación.
- Control automático de versiones para trazabilidad de experimentos.
- Integración con diversas aplicaciones de terceros como Microsoft Excel, Jupyter, SQL, Pandas, Darknet, Pytorch, etc. Agnóstico de la plataforma.

En resumen, Valohai es una alternativa con licencia comercial a MLFlow, que además de hacer todo lo que hace MLFlow, permite a los desarrolladores gestionar la infraestructura en donde se despliega todo el entorno tanto de desarrollo como de producción, teniendo la posibilidad de automatizar el inicio o apagado de máquinas en la nube para entrenamiento.

3.4.6. Plataformas Cloud

Hasta ahora, en este trabajo se ha hablado en todo momento de herramientas que un desarrollador o cualquier empresa puede utilizar de forma interna, administrando ellos la solución y montando una arquitectura propia que ayude a mejorar todo el proceso de desarrollo y puesta en producción de un software de Inteligencia Artificial.

Aunque las plataformas cloud no entran dentro del ámbito de estudio de este trabajo, sí considero importante al menos mencionar algunas de las soluciones existentes en el mercado para poder tener una imagen completa de qué es lo que ofrecen empresas como Amazon, Microsoft o Google a la hora de desplegar procesos MLOps en cloud.

3.4.6.1. Amazon AWS

Página Oficial: https://aws.amazon.com/es/sagemaker/mlops/

En primer lugar, la plataforma de Amazon brinda su solución MLOps a través de Sagemaker, su herramienta de Inteligencia Artificial. En la figura 3.17 se observa el pipeline propuesto para este cometido.

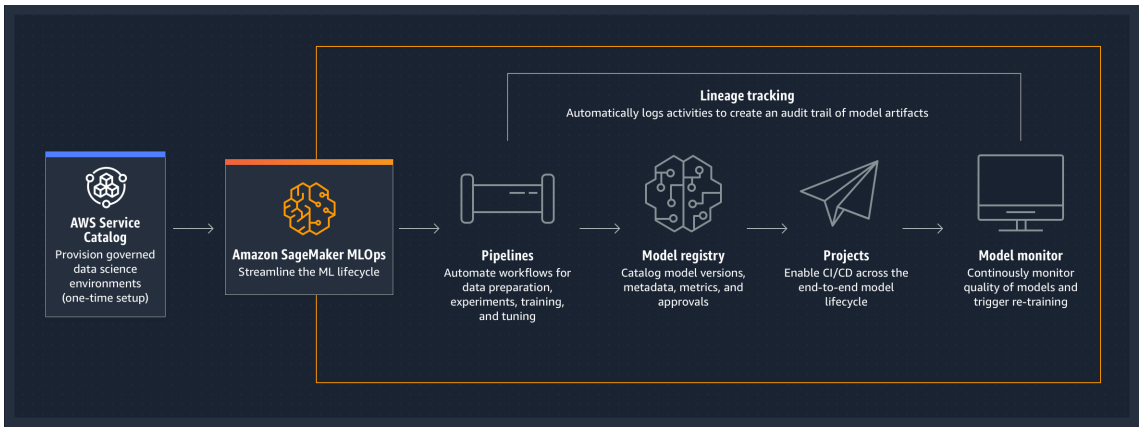

Figura 3.17: Pipeline MLOps de AWS Sagemaker

Las principales características expuestas por AWS son:

- Contrucción de ciclos de integración y despliegue continuos para reducir la complejidad a la hora de gestionar modelos.
- Automatización de flujosd de Machine Learning.

- Monitorización del rendimiento de los modelos de Machine Learning para detectar automáticamente sesgo, cambios del entorno o degradación en general.
- Seguimiento automático del código, los datasets, los artefactos generados y en general de todos los pasos en el pipeline.

3.4.6.2. Azure

Página Oficial: https://azure.microsoft.com/es-es/services/machine-learning/mlops/#features

Azure, la plataforma gestionada por Microsoft, tiene directamente integradas las operaciones MLOps en su herramienta cloud llamada Azure Machine Learning, de la cual destacan las siguientes funcionalidades:

- Contrucción de ciclos de integración y despliegue continuos para reducir la complejidad a la hora de gestionar modelos.
- Automatización de flujosd de Machine Learning.
- Monitorización del rendimiento de los modelos de Machine Learning para detectar automáticamente sesgo, cambios del entorno o degradación en general.
- Seguimiento automático del código, los datasets, los artefactos generados y en general de todos los pasos en el pipeline.

En la figura 3.18 se observa la arquitectura propuesta por Microsoft para MLOps en Azure.

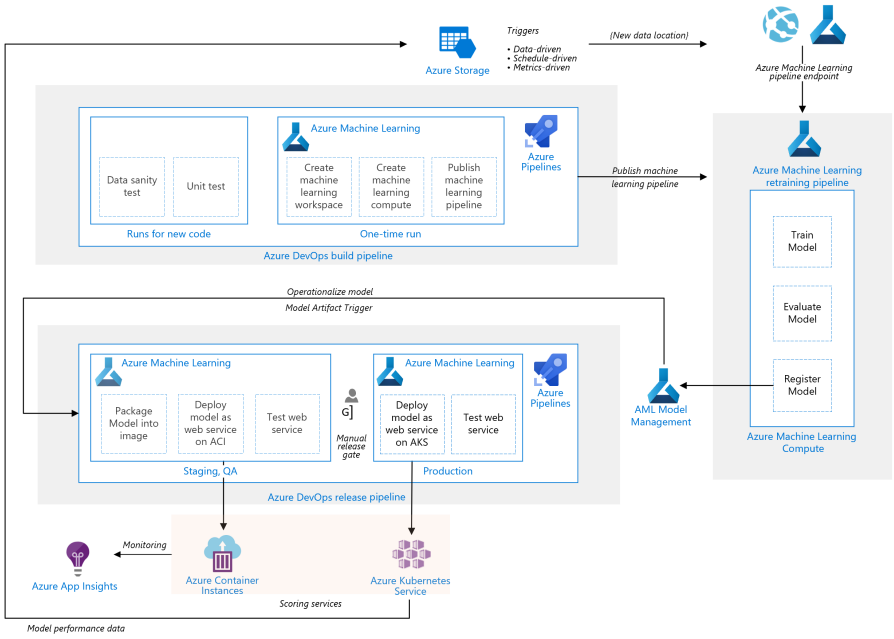

Figura 3.18: Arquitectura de Azure MLOps

3.4.6.3. Google Cloud Platform

Página Oficial: https://cloud.google.com/products/ai?hl=es

La propuesta de Google es algo diferente, ya que en general el ecosistema de Google está enfocado de una forma diferente a las infraestructuras de Amazon o Azure. Google dota a los desarrolladores de diferentes herramientas para hacer proyectos de Inteligencia Artificial, desde herramientas simples como Google Colab (implementación propia de Juypter Notebooks), hasta otras herramientas más específicas como Video AI o Translation AI, destinados únicamente a un tipo concreto de proyectos.

En esa línea, Google Cloud dispone de una guía para entender MLOps, la cual se puede encontrar en el siguiente link. cloud.google.es.

Tras haber expuesto todos los pasos hasta llegar al máximo grado de madurez de MLOps, en el artículo se exponen diferentes alternativas para aplicar estas técnicas en Google Cloud, como por ejemplo canalizar el CI/CD mediante Kubeflow y Cloud Build, o tener entrega continua con GitOps.

La mejor forma de comprender cómo se gestiona MLOps en Google Cloud es este vídeo, subido por Google Cloud Tech, que explica las herramientas y los pipelines necesarios para aprovechar todo el potencial MLOps en Google Cloud: https://www.youtube.com/watch?v=6gdrwFMaEZ0

3.5. Diseño e implementación del sistema demostrador

El objetivo de este TFM es sobretodo el desarrollo de una guía de técnicas y herramientas para profundizar sobre la idea de MLOps, una idea novedosa que aún no está del todo implementada en prácticamente ninguna compañía, pero para cerrar el trabajo y ejemplificar las capacidades de estas técnicas, en este apartado se trata de diseñar una arquitectura sencilla que ponga de manifiesto algunas ventajas expuestas a lo largo del trabajo.

El objetivo de esta sección es conseguir un pipeline completo con MLOps que garantice que ya sea en un desarrollo pequeño o en una gran compañía, es posible crear un flujo de trabajo eficaz que garantice desde el primer momento la automatización de varios elementos del sistema, principalmente a la hora de facilitar la puesta en producción del modelo.

3.5.1. Arquitectura y Herramientas Finales del Sistema

Tomando como referencia la comparativa de herramientas del apartado anterior, en este apartado se ha diseñado una arquitectura de referencia para poder realizar pruebas a pequeña escala de todas las ventajas que supone aplicar las técnicas MLOps en cualquier proyecto, tenga el tamaño que tenga.

Así, la arquitectura desplegada consta de los siguientes elementos:

- Para el control de versiones de código se ha utilizado **Git**, con **GitLab** como servidor local.
- Para el seguimiento de los experimentos y modelos generados se utilizará **MLFlow**
- Para el API REST del modelo se ha utilizado **Flask**
- Para empaquetar la solución en un contenedor se ha utilizado **Docker**
- Para automatizar el proceso de empaquetado y despliegue se utiliza **Jenkins**

El diagrama final de la arquitectura se puede ver en la figura 3.19:

3.5.2. Descripción y resultados del caso de uso

El sistema planteado consiste en desarrollar una Red Neuronal Convolucional (CNN) que reciba mediante un portal Web el dibujo de un número escrito a mano del 0 al 9 y devuelva como

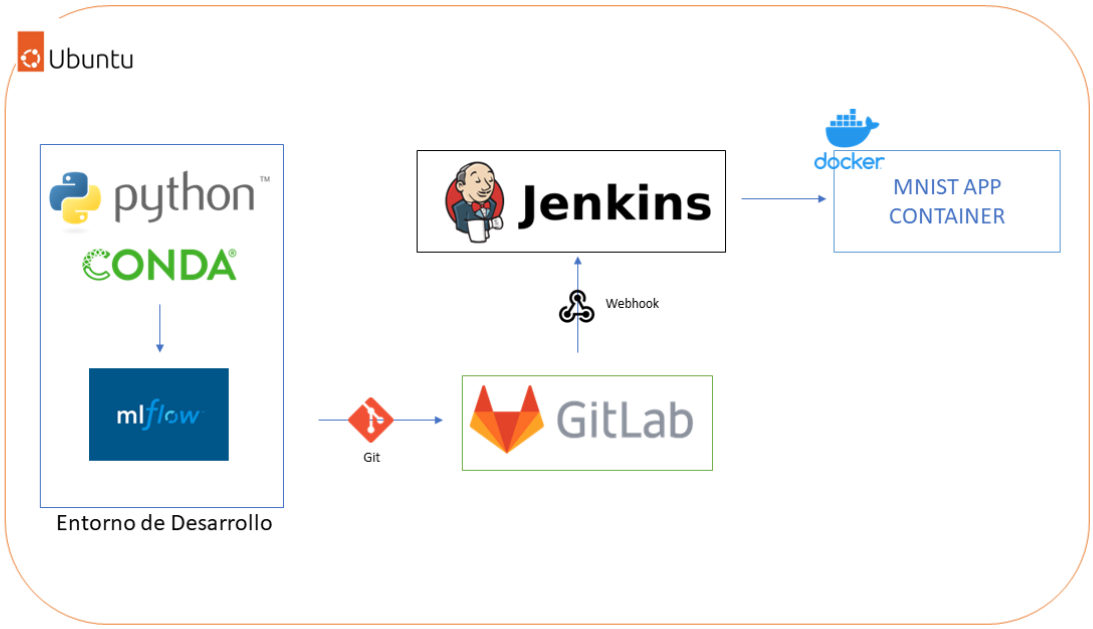

Figura 3.19: Arquitectura del Caso de Uso

resultado el número, es decir, el algoritmo deberá decidir qué número es el que se ha dibujado.

Para identificar el valor que puede aportar MLFlow como herramienta MLOps, se realizan diferentes entrenamientos, quitando distintos números y añadiéndolos posteriormente para poder comparar varios modelos diferentes, con distintas métricas. En los primeros entrenamientos, se eliminan algunos números al conjunto de entrenamiento, lo que provoca que la precisión en entrenamiento sea alta, pero disminuya mucho en validación. Es lógico que si entrenamos al modelo sin un número, será incapaz de predecir dicho valor.

Progresivamente, se realizan entrenamientos añadiendo diferentes números, hasta conseguir una precisión mejor. Esta precisión se consigue entrenando con todos los números menos el 0, ya que este número será entrenado después para simular una actualización en pre-producción.

En la figura 3.20 se observa la lista de modelos entrenados en lo que simula ser la fase de desarrollo, con las diferentes columnas de métricas indicando los distintos valores de precisión, pérdida y precisión sobre el set de validación.

Además, en la figura 3.21 se observa una gráfica comparativa entre la precisión sobre el conjunto de validación de los diferentes modelos.

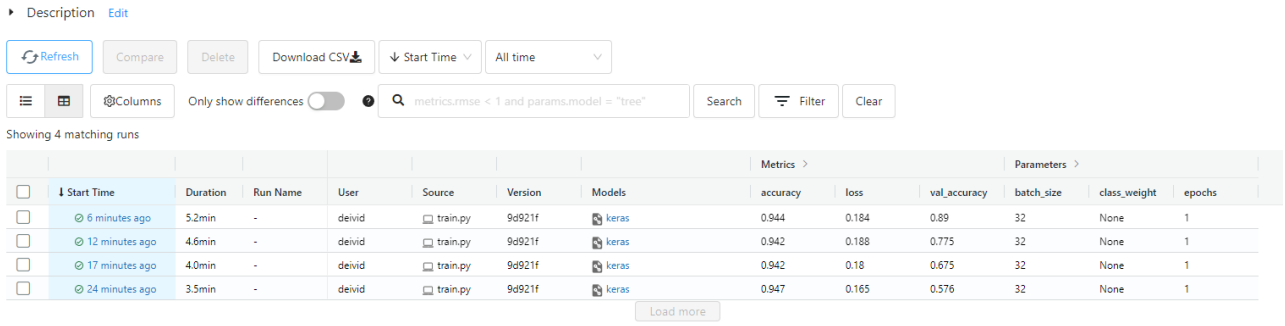

Figura 3.20: Tracking de modelos con MLFLow

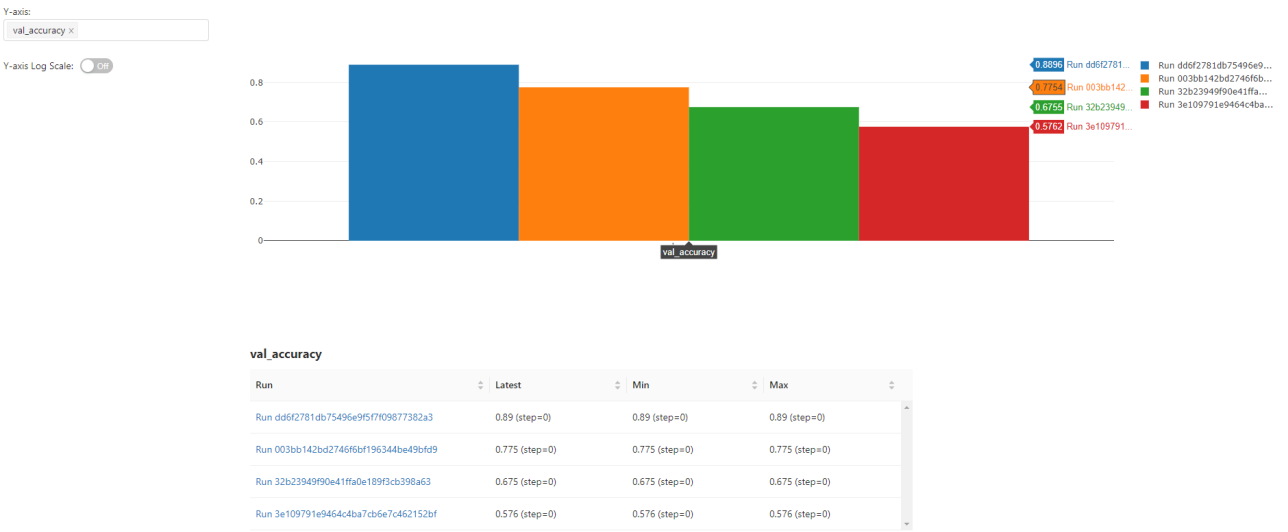

Figura 3.21: Comparativa de la precisión sobre el set de Validación

Vistos los resultados, se decide desplegar en producción el modelo con mayor precisión, es decir, el último que se ha entrenado. Para ello, se utiliza un servidor GitLab instalado sobre Ubuntu como repositorio de código remoto, y Jenkins para automatizar el pipeline de subida. Jenkins, mediante el plugin de GitLab y los Webhooks, tiene la capacidad de detectar cuándo se ha producido una subida a una determinada rama, en este caso, la rama master. Cuando detecta la subida, procede a ejecutar una serie de pasos definidos en el Jenkinsfile. Como el modelo ya está entrenado y subido a GitLab, con Jenkins queremos conseguir que se edspliegue en un contenedor la solución y se ponga a disposición de los usuarios. En este caso de uso, los pasos definidos son:

1. Se compila la imagen Docker mediante el Dockerfile. La definición de este Dockerfile se especifica en el anexo A
2. Se para el contenedor anterior de la solución en caso de existir y se despliega el nuevo con el modelo actualizado.

En la figura 3.22 se observa una captura de pantalla de la web desplegada, incapaz de predecir correctamente el número 0.

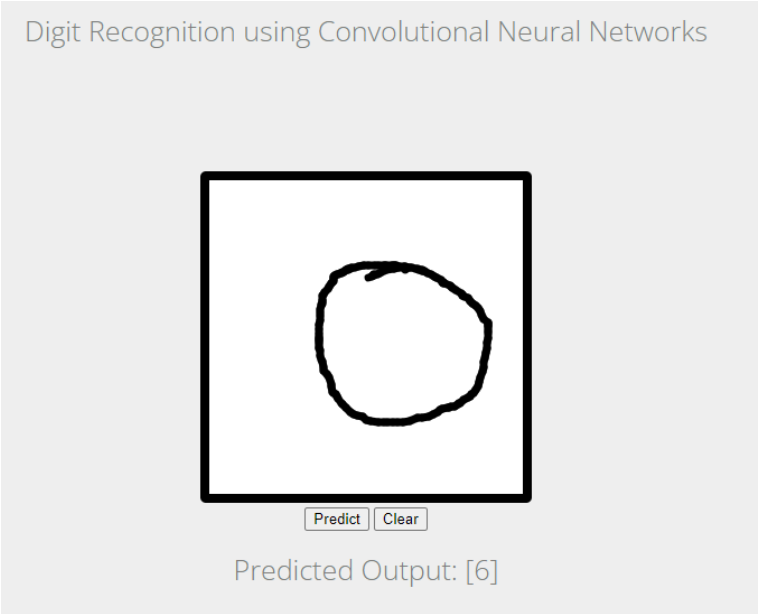

Figura 3.22: App en pre-producción

Además, en la figura 3.23 se observa el repositorio local de Gitlab con la solución.

Por último, en la figura 3.24 se puede ver la ejecución generada en Jenkins.

Con el modelo en pre-producción, **muchos usuarios comienzan a reportar que este modelo no funciona bien**, y que cada vez que dibujan un 0 el resultado no es el esperado. Sabiendo esto, se lanza un reentrenamiento con el dataset ampliado con el número 0, obteniendo un modelo con mayor precisión sobre el set de validación.

Con el modelo entrenado, se lanza la puesta en pre-producción automática del nuevo modelo, con lo que el paso de desarrollo a producción es prácticamente instantáneo (lo que tarda en subirse a Gitlab y ejecutarse en Jenkins, pero sin ninguna intervención por parte del desarrollador).

En la figura 3.25 se observa la comparativa de todos los modelos desarrollados, con el último con mayor precisión.

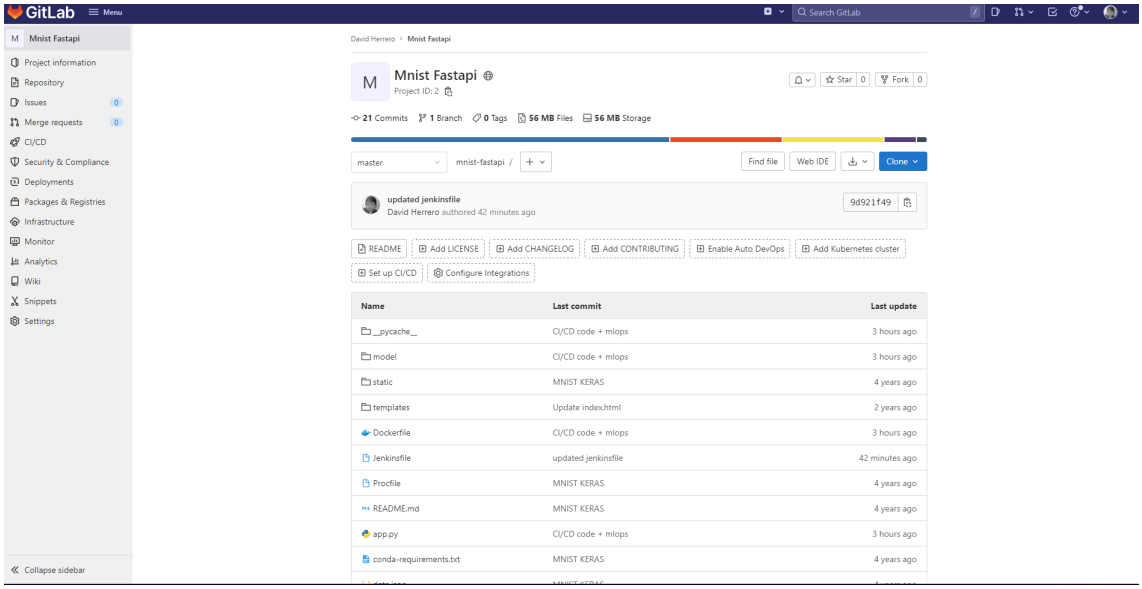

Figura 3.23: Código subido a Gitlab

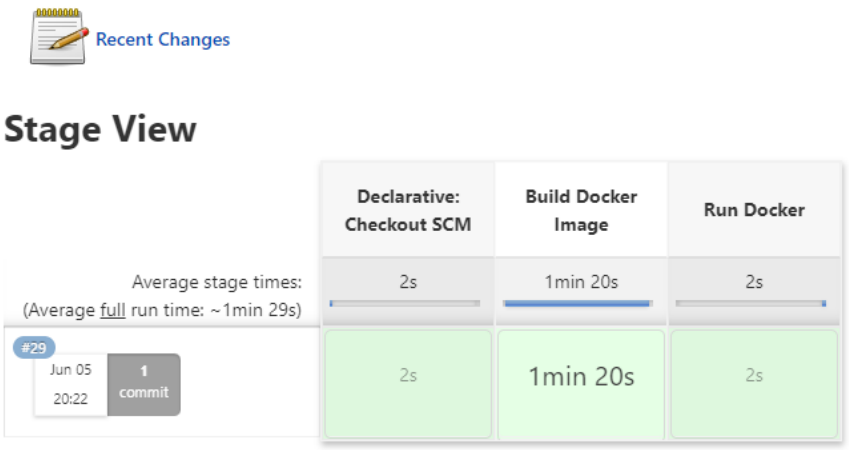

Figura 3.24: Ejecución en Jenkins

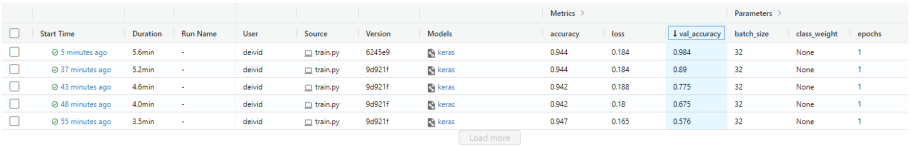

Figura 3.25: Modelos vistos en MLFlow

Una vez generado este modelo, se sube el código a Gitlab mediante push a la rama master, y automáticamente se lanza un nuevo trabajo en Jenkins, que despliega de nuevo la aplicación en un contenedor y elimina el anterior.

La nueva aplicación es capaz de detectar el número 0 sin ningún problema, como se observa en la figura 3.26

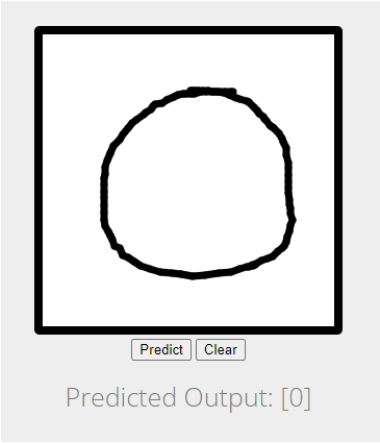

Figura 3.26: App detectando el número 0

3.5.3. Desarrollo del sistema

Debido a la complejidad de la instalación e implementación del sistema, todo el código desarrollado, las referencias para instalación de las herramientas, así como la forma de interconectar las diferentes herramientas, se expone en el apéndice A

Capítulo 4

Conclusiones y Resultados

A lo largo del TFM, se han expuesto diferentes herramientas, metodologías, arquitecturas y, en general, formas de implementar satisfactoriamente una gestión de Machine Learning Operations óptima y que ayude a los desarrolladores, data scientist y organizaciones a poner en producción y acelerar el desarrollo de los proyectos de Inteligencia Artificial, para que por fin este tipo de proyectos aporten valor real tanto a los usuarios como a las compañías.

4.1. Métricas clave

Aunque en este apartado y en este TFM sea complicado establecer un porcentaje concreto de mejora en el desarrollo, existen cuatro métricas claves para entender los beneficios de la aplicación de estas herramientas y técnicas:

1. **Plazo para modificaciones**: Una de las principales ventajas de las técnicas MLOps es el plazo necesario para realizar modificaciones en el código o entrenar modelos con datos nuevos. Con la automatización expuesta a lo largo del TFM y demostrada con el sistema de prueba, se ha reducido drásticamente el plazo de modificaciones, teniendo mucho más control sobre todos los experimentos y desplegando automáticamente la nueva versión.

2. **Tasa de errores por modificaciones**: El porcentaje de cambios realizados en el código que requieren una corrección inmediata tras haber pasado a producción. Al tener un proceso automático de integración y despliegue, estos errores se localizan mucho más rápido, y, aplicando esta tasa a los modelos de Inteligencia Artificial, gracias a la trazabilidad de experimentos y las métricas de precisión se puede ver de forma rápida cuándo un modelo

funciona mejor que otro.

3. **Frecuencia de Implementación**: Es fundamental saber con qué frecuencia se implementa código nuevo en la fase de producción para poder medir el éxito de MLOps. Muchos profesionales utilizan el término .^{en}trega"para referirse a los cambios de código que se publican en un entorno de ensayo de preproducción y reservan el concepto ïmplementación"para aquellos cambios de código que se publican en producción.

4. **Tiempo medio de recuperación (MTTR)**: Indica el tiempo medio que se tarda en recuperar el sistema después de un fallo que provoque su caída.

Con estas cuatro métricas, se pone en evidencia que las técnicas MLOps son eficaces a la hora de mejorar el tiempo desde el comienzo del desarrollo hasta la puesta en producción, y posteriormente en la fase de mantenimiento y monitorización del modelo en producción, ya que todos los tiempos se reducen drásticamente y el sistema se vuelve mucho más mantenible y sostenible en el tiempo.

4.2. Líneas Futuras

Una de las mayores preocupaciones de MLOps y la principal línea de mejora de este TFM y en general de todos los ecosistemas MLOps actuales es el reentrenamiento automatizado y la monitorización de los modelos en producción. Aunque monitorizar un modelo parece algo sencillo, es realmente difícil saber cuándo un modelo no está dando las predicciones correctas sin supervisarlo un humano. En el caso de uso desarrollado, por ejemplo, solamente una persona que use el sistema sabe cuándo un número está mal predicho, pero un sistema de monitorización automático no puede saberlo de ninguna manera sin feedback de los usuarios.

En este sentido, existen algunas aproximaciones combinando estas técnicas con algoritmos de auto etiquetado, comparando la etiqueta con la predicción, o algoritmos de detección de anomalías a la hora de predecir resultados numéricos, pero es una línea que realmente tiene que desarrollarse para poder llegar a una automatización plena sin prácticamente supervisión.

Bibliografía

[1] Simple devops project-3 — devops project with git, jenkins and docker on aws — cicd on containers. https://www.youtube.com/watch?v=nMLQgXf8tZ0, Oct 2018.

[2] MLOps.org. Ml-ops.org: Motivation. https://ml-ops.org/content/motivation, Dec 2021.

[3] Rosa Fernández. Tipologia de las inversiones en inteligencia artificial 2017-2020. es.statista.com, Ago 2021.

[4] Gartner says nearly half of cios are planning to deploy artificial intelligence. gartner.com, Feb 2018.

[5] Shervin Khodabandeh Sam Ransbotham. Winning with ai. sloanreview.mit.edu, Oct 2019.

[6] VB Staff. Why do 87 % of data science projects never make it into production? venturebeat.com, Jul 2019.

[7] Edix Digital Workers. Programación estructurada. edix.com/, Jul 2021.

[8] David Hemmendinger. Algol. britannica.com, Feb 2018.

[9] Edsger W. Dijkstra. Letters to the editor: Go to statement considered harmful. https://doi.org/10.1145/362929.362947, mar 1968.

[10] What the waterfall project management methodology can (and can't) do for you. lucidchart.com, Sep 2018.

[11] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development. https://agilemanifesto.org/, 2001.

[12] Ian Buchanan. Historia de devops. https://www.atlassian.com/es/devops/what-is-devops/history-of-devops, 2020.

[13] Atlassian. Atlassian survey 2020 - devops trends. https://www.atlassian.com/es/whitepapers/devops-survey-2020, 2020.

[14] The Linux Foundation. 10 years of git: An interview with git creator linus torvalds. linuxfoundation.org, Aug 2017.

[15] Stack Overflow. Stack overflow developer survey 2021. insights.stackoverflow.com, 2021.

[16] Free BSD. Chapter 15. jails. https://docs.freebsd.org/en/books/handbook/jails/, 2022.

[17] Kohsuke Kawaguchi. Hudson. Webarchive:java.net, 2007.

[18] Algorithmia. info.algorithmia.com, 2021.

[19] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. https://proceedings.neurips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf, 2015.

[20] Kubernetes. Production-grade container orchestration. https://kubernetes.io/, 2022.

[21] Anaconda. State of data science 2021. https://www.anaconda.com/state-of-data-science-2021, 2021.

[22] Google Cloud. Mlops: Continuous delivery and automation pipelines in machine learning. cloud.google.com, 2022.

[23] Microsoft Azure. Modelo de madurez de operaciones de machine learning. docs.microsoft.com, Junio 2020.

Apéndice A

Instalación e Implementación de la Demo

En este apéndice se recopila toda la documentación utilizada para instalar diferentes herramientas en el entorno de demostración. Además, se explica el código desarrollado y la interconexión entre las diferentes herramientas utilizadas para el entorno.

A.1. Instalación de Ubuntu Server en VirtualBox

La instalación de VirtualBox es un proceso gráfico y sencillo. Un ejemplo de instalación se puede encontrar en la URL https://support.academicsoftware.eu/hc/es/articles/360006725277-C%C3%B3mo-instalar-Oracle-VM-VirtualBox

Una vez instalado VirtualBox, se instala Ubuntu Server, siguiendo la documentación de Canonical https://ubuntu.com/tutorials/install-ubuntu-server#1-overview

A.2. Instalación de Jenkins en Ubuntu Server

Para instalar Jenkins en Ubuntu Server, lo primero que se necesita es Java, concretamente el JRE 8 o superior. Para instalar Java JRE 8, se ejecuta el siguiente comando:

```
1 sudo apt-get install openjdk-8-jre
```

Una vez instalado Java, se procede a la instalación de Jenkins, para la cual se ha utilizado

como referencia la siguiente guía: https://www.digitalocean.com/community/tutorials/how-to-install-jenkins-on-ubuntu-20-04-es

A.3. Instalación de Gitlab en Ubuntu Server

La guía de instalación de Gitlab se puede encontrar en su página oficial: https://about.gitlab.com/install/#ubuntu

A.4. Código de la solución

Todo el código de la solución se puede encontrar en el siguiente repositorio de github: https://github.com/Deividhp13/mnist-flask, siendo un fork del repo alojado en https://github.com/akashdeepjassal/mnist-flask

A.4.1. Modelo IA

El modelo de IA se trata de una red neuronal convolucional implementada con Keras y Tensorflow. Las capas de la red neuronal se pueden ver en el siguiente código:

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

El dataset MNIST es obtenido a través del API de Keras, concretamente mediante el siguiente código:

```
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

A.4.2. Jenkinsfile

El Jenkinsfile desarrollado contiene lo siguiente:

```
pipeline {
    agent any
    stages {
        stage('Build Docker Image') {
            steps {
                sh "sudo docker build -t mnist-fastapi:latest ."
            }
        }
        stage('Run Docker'){
            steps {
                sh """
                sudo docker stop mnist || true
                sudo docker rm mnist || true
                sudo docker run -d -p 7000:7000 --name mnist mnist-fastapi
                """
            }
        }
    }
}
```

En él, se define un pipeline en el que se compila la imagen docker, adjuntando todos los archivos generados en el desarrollo, para posteriormente parar el contenedor antiguo en caso de estar en ejecución y desplegar el nuevo sobre el puerto 7000.

A.4.3. Dockerfile

El archivo DockerFile define el contenedor que se despliega en producción, que contiene todas las dependencias y el modelo entrenado. Para este Dockerfile se utiliza como imagen base miniconda3 con python 3.6, para poder instalar todos las librerías necesarias y desplegar. El código del dockerfile es el siguiente:

```
FROM continuumio/miniconda3:4.7.12
COPY . /mnist-fastapi
WORKDIR /mnist-fastapi
RUN conda install --yes --file requirements.txt
CMD ["python", "app.py"]
```

A.4.4. Flask API

Por último, el código desarrollado en Flask permite la interacción de los usuarios con el modelo. Contiene dos rutas clave, en primer lugar la ruta raíz, que muestra el índice de la web, y en segundo lugar la ruta predict, que permite mandar imágenes al modelo para realizar la inferencia.

Ambos métodos se especifican en el siguiente código:

```
@app.route('/')
def index():
    return render_template("index.html")

@app.route('/predict/', methods=['GET','POST'])
def predict():
    # get data from drawing canvas and save as image
    parseImage(request.get_data())

    # read parsed image back in 8-bit, black and white mode (L)
    x = imread('output.png', mode='L')
    x = np.invert(x)
    x = imresize(x,(28,28))

    # reshape image data for use in neural network
    x = x.reshape(1,28,28,1)
    with graph.as_default():
        set_session(sess)
        out = model.predict(x)
        print(out)
        print(np.argmax(out, axis=1))
        response = np.array_str(np.argmax(out, axis=1))
        return response
```

A.5. Interconexión de GitLab con Jenkins para automatizar despliegues

La automatización de Jenkins y Gitlab se ha realizado mediante un Webhook. Los pasos a seguir han sido:

1. A través del panel de administración de Jenkins, instalar el plugin de GitLab.

Administrar Jenkins →Administrar Plugins →Todos los Plugins →GitLab

2. Obtener un Access Token en Gitlab, para ello, entrar en User Settings →Access Token
3. Poner las credenciales y el Access Token en Jenkins, en Administrar Jenkins, sección de GitLab.
4. Crear un proyecto de tipo Pipeline en Jenkins, con la compilación automática en PUSH activada. Esta opción se puede ver en la figura A.1
5. Crear el webhook asociado al proyecto en GitLab con la información de Jenkins, el cual se puede ver en la figura A.2, para eventos PUSH.

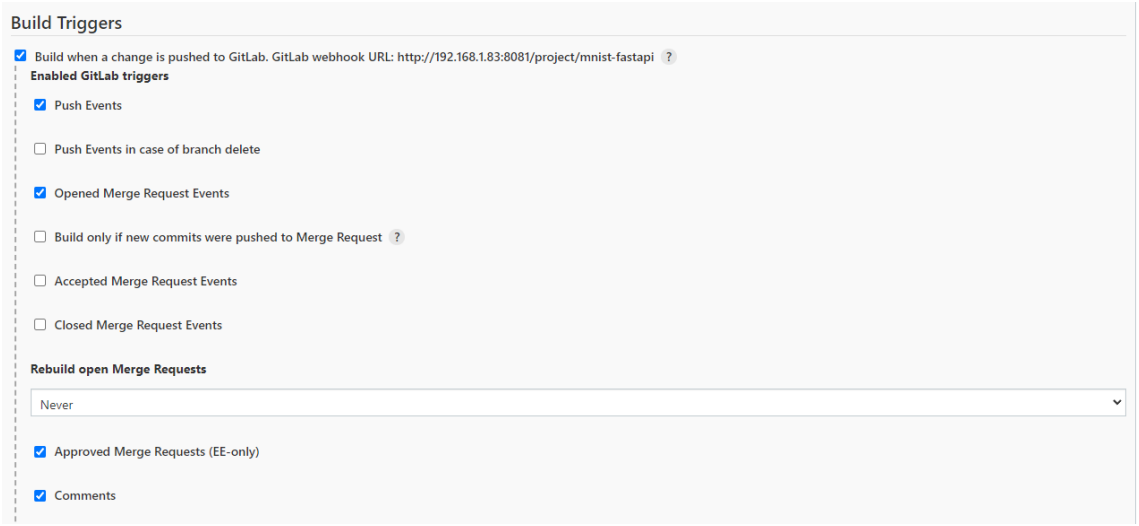

Figura A.1: Proyecto Pipeline con GitLab en Jenkins

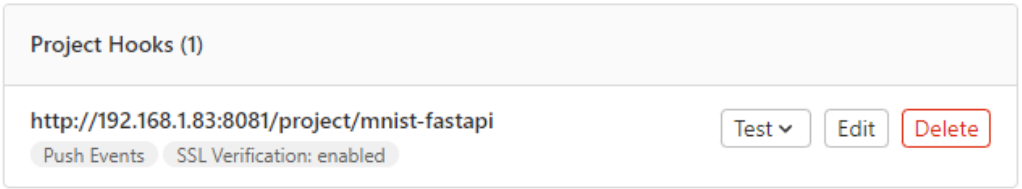

Figura A.2: Webhook de automatización Jenkins y GitLab