UNIVERSIDAD DE LA RIOJA

# TRABAJO FIN DE ESTUDIOS

Título

## Machine Learning Operations (MLOps): contexto actual y tendencias futuras


Autor/es

MILLÁN SANTAMARÍA SACRISTÁN

Director/es

FRANCISCO JAVIER MARTÍNEZ DE PISÓN ASCACÍBAR

Facultad

Escuela de Máster y Doctorado de la Universidad de La Rioja


Titulación

Máster Universitario en Dirección de Proyectos

Departamento

INGENIERÍA MECÁNICA

Curso académico

2022-23



publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

# Trabajo de Fin de Máster

# Machine Learning Operations (MLOps): contexto actual y tendencias futuras

# Machine Learning Operations (MLOps): current context and future trends

Autor : ***Santamaría Sacristán, Millán***

Tutor: Martínez de Pisón Ascacibar, Fco. Javier

**MÁSTER:**
**Dirección de Proyectos**

**Escuela de Máster y Doctorado**

**UNIVERSIDAD DE LA RIOJA**

**AÑO ACADÉMICO: 2022/2023**

# Contenido

## Resumen

Las metodologías MLOps buscan incluir los procesos de aprendizaje automático y ciencia de datos en la cadena de desarrollo y operaciones para hacer que los proyectos de implementación del Machine Learning sean más confiables y productivos. El objetivo del TFM consistirá en analizar el contexto actual describiendo las diversas metodologías existentes, analizando las fortalezas y debilidades, puntos en común y diferencias, buenas prácticas de cada una, etc. Finalmente, se intentará realizar una proyección de como evolucionarán éstas en el futuro. Este trabajo pretende ser una guía en la que puedan apoyarse todas aquellas personas que necesiten aplicar esta metodología.

## Abstract

MLOps methodologies seek to include machine learning and data science processes in the development and operations chain to make Machine Learning implementation projects more reliable and productive. The objective of the TFM will be to analyze the current context by describing the various existing methodologies, analyzing the strengths and weaknesses, commonalities and differences, best practices of each one, etc. Finally, an attempt will be made to make a projection of how they will evolve in the future. This work is intended as a guide for all those who need to apply this methodology.

# 1. Introducción

## 1.1. Contexto y antecedentes

En los últimos años, el *machine learning* (ML) y la inteligencia artificial (IA) se han convertido en disciplinas cada vez más populares. Esto se debe principalmente al considerable aumento en la capacidad de cálculo y almacenamiento de las computadoras. Por citar algunos ejemplos populares, el modelo de aprendizaje automático *Google DeepMind AlphaGo* (https://deepmind.com/research/case-studies/alphago-the-story-so-far), que logró vencer al campeón del juego de mesa *GO*, considerado históricamente como un desafío para la IA, o un modelo desarrollado en Stanford para la detección del cáncer de piel, que alcanzó una precisión comparable a la de expertos médicos (https://www.nature.com/articles/nature21056).

Además del aumento en la capacidad de cálculo, el costo de los componentes ha disminuido significativamente, lo que ha permitido que estos avances estén al alcance de toda la sociedad. Cada vez encontramos más aplicaciones de ML en nuestra vida cotidiana, desde *smartphones* hasta modelos populares como *ChatGPT*[1], que con su simplicidad de uso han facilitado el acceso masivo a un potente modelo de ML.

La necesidad de emplear modelos para predecir y procesar datos se ha extendido a todos los sectores de la industria. Las empresas cada vez generan mayor volumen de datos, muchos de los cuales se pierden o se almacenan sin saber cómo aprovecharlos. Sin embargo, los avances en el procesamiento de datos y la IA en los últimos años han hecho que numerosas empresas reconozcan el valor de estos datos. A través de ellos, se puede extraer un gran conocimiento que puede brindar ventajas competitivas y ayudar a optimizar procesos, lo que se traduce en un aumento de beneficios tanto por la mejora de los procedimientos y productos, como por su optimización.

Una vez que se ha desarrollado un modelo óptimo, no se puede dar por terminado su desarrollo. Un modelo por sí solo no aporta valor, se necesitan datos para nutrir el modelo y realizar inferencia sobre ellos. El modelo es solo una parte de un todo que incluye al propio modelo, los datos y la plataforma en la que se utilizará.

En el mundo de la informática, especialmente en los últimos años, donde los cambios rápidos y constantes son una realidad, la eficiencia en la industrialización es crucial. Si no logramos cumplir con los plazos estimados, el modelo que tanto esfuerzo costó desarrollar y que debería brindar una ventaja competitiva podría volverse obsoleto en el nuevo contexto, haciendo que todos los esfuerzos en su desarrollo hayan sido en vano. Por lo tanto, la industrialización de modelos mediante la aplicación de MLOps se convierte en un pilar fundamental para obtener el máximo beneficio de la IA en el contexto empresarial

[1] https://openai.com/chatgpt

actual. La combinación de conocimientos técnicos sólidos, la planificación eficiente y la gestión adecuada del ciclo de vida de los modelos son elementos esenciales para lograr el éxito en esta era de transformación digital acelerada.

Teniendo en cuenta los factores mencionados anteriormente, es necesario incorporar profesionales con conocimientos de MLOps en los equipos de científicos de datos de las empresas con el fin de industrializar los modelos y asegurar su aporte de valor. Estos expertos son capaces de anticipar riesgos y resolver problemas en la implementación de modelos en entornos del mundo real. Además, cuentan con un profundo conocimiento de los algoritmos de IA, así como habilidades en gestión de proyectos, ingeniería de software y seguridad de datos.

Descuidar estos aspectos puede llevar al fracaso, incluso si el modelo es bueno. MLOps, como disciplina, abarca técnicas para la industrialización eficiente y robusta de los modelos de IA. En este trabajo, ampliaremos la definición de MLOps a lo largo del estudio.

## 1.2. Objetivos

El principal objetivo de este trabajo de fin de máster es realizar un estudio sobre las metodologías actuales y futuras en el ámbito de MLOps. Lo que se busca tras la realización del estudio es obtener las características principales de los diversos enfoques MLOps actuales y los que se proponen a futuro, buscando puntos en común y su motivo, así como las diferencias y sus justificaciones.

En él se tratará de analizar esta nueva corriente de la IA haciendo hincapié en los beneficios que puede aportar la implementación de esta metodología explorando los diferentes enfoques y herramientas que nos permitan adoptar una arquitectura con filosofía MLOps.

En último lugar, se planteará una comparativa entre las herramientas actuales del mercado, tanto gratuitas como de pago, así como entre los enfoque y arquitecturas propuestas por los principales proveedores de servicios en la nube como Google, Amazon o Microsoft. También se analizarán las tendencias del mercado centrándose en los proveedores en la nube de forma que nos permitan ver cómo evolucionará esta metodología en el futuro desde su punto de vista.

# 2. MLOps, Operaciones en Aprendizaje Automático

## 2.1. Problemática en los Sistemas de Machine Learning

Una idea comúnmente aceptada dentro de la comunidad de aprendizaje automático (ML) es que el proceso de desarrollo e implementación de sistemas ML es rápido y económico, sin embargo, el mantenimiento de estos sistemas a lo largo del tiempo resulta ser un desafío complicado y costoso.

Esta dicotomía se puede entender a través de la metáfora de la deuda técnica[2]. No todas las deudas son malas, pero todas las deudas deben ser pagadas. La deuda técnica se puede paliar refactorizando el código, mejorando las pruebas unitarias, eliminando el código muerto, reduciendo las dependencias, documentando, etc. El objetivo no es agregar nuevas funcionalidades, sino permitir mejoras futuras, reducir errores y mejorar la capacidad de mantenimiento. Ignorar estas tareas puede derivar en elevados costos. Además de esta deuda, también existe la deuda oculta que es peligrosa porque se acumula en silencio.

Otro de los puntos a tener en cuenta es que el uso de herramientas para la creación rápida de soluciones ML puede resultar a priori económico, pero a la larga se puede incurrir en elevados costes derivados de su mantenimiento y deuda técnica.

En este apartado se analizará la capacidad de los sistemas ML para incurrir en deuda técnica, ya que tienen todos los problemas de mantenimiento del código tradicional más un conjunto adicional de problemas específicos de ML. Esta deuda puede ser difícil de detectar porque existe a nivel de sistema en lugar de a nivel de código. Los datos pueden influir sutilmente en el comportamiento del sistema de aprendizaje automático, lo que puede corromper o invalidar las abstracciones y límites tradicionales. Los métodos comunes para resolver la deuda técnica a nivel de código no son suficientes para abordar la deuda técnica específica de ML a nivel del sistema.

### 2.1.1. Los modelos complejos erosionan los límites

Se ha comprobado a través de la práctica tradicional en la ingeniería de software que establecer límites claros de abstracción como encapsulación y diseño modular de código permiten crear un código más fácilmente mantenible, siendo más sencillo poder realizar cambios y mejoras aisladas. Estos límites estrictos de abstracción permiten representar la estabilidad y coherencia lógica de las entradas y salidas de un componente específico.

[2] **Deuda técnica**: metáfora introducida por Ward Cunningham en 1992 para ayudar a razonar sobre los costos a largo plazo incurridos al moverse rápidamente en la ingeniería de software. Refleja el costo implícito del retrabajo adicional causado por elegir una solución fácil en lugar de utilizar un enfoque que llevaría más tiempo en su desarrollo e implementación.

Lamentablemente, aplicar estos límites de abstracción estrictos en los sistemas de aprendizaje automático es un desafío debido a su naturaleza. Es complicado prever un comportamiento específico y, en muchos casos, el comportamiento deseado no se puede expresar adecuadamente en la lógica del software sin recurrir a ML y fuentes externas de datos. Por ejemplo, resolver un laberinto puede lograrse con técnicas de software, pero identificar qué marco contiene el ratón o el queso requiere de técnicas de aprendizaje automático, como se ilustra en la *Figura 1*.

El mundo real no siempre encaja en una estructura bien organizada, por lo que se analizarán diferentes formas en que la erosión de los límites puede aumentar significativamente la "deuda técnica" en los sistemas de ML.

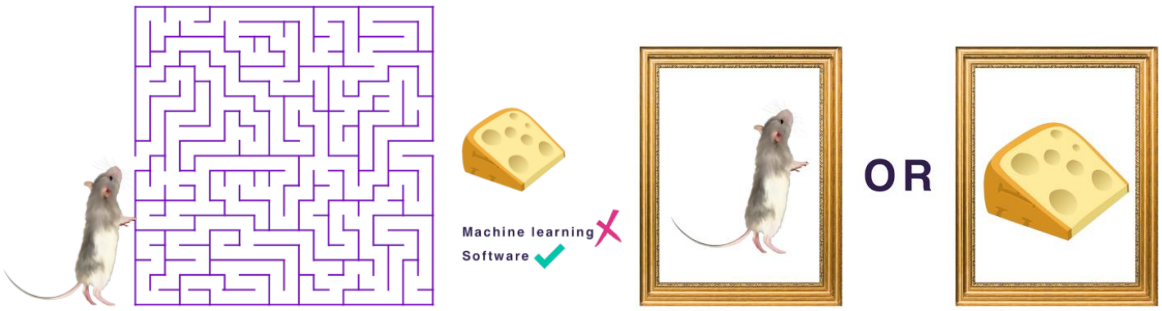



**Figura 1. Diferencia entre software y ML**

En los sistemas de aprendizaje automático, existe un problema de enredo de las entradas, lo que dificulta aislar las mejoras individuales. Esto se debe a que cualquier cambio en una variable puede afectar a las otras variables. Incluso agregar o eliminar una variable puede no producir resultados diferentes, ya que todas las entradas están interconectadas como vemos en la *Figura 2*. Este fenómeno se conoce como el principio CACE (*change anything can chage everything*).

Una forma de abordar este problema es mediante el uso de modelos *ensembled*, que consisten en combinar varios modelos individuales. Sin embargo, los conjuntos de modelos solo funcionan bien si los errores de los componentes no están correlacionados. De lo contrario, mejorar un modelo individual puede empeorar la precisión global debido a la fuerte correlación entre los errores.

Otra alternativa es enfocarse en la identificación de cambios en el comportamiento de la predicción en tiempo real. Una opción propuesta es utilizar herramientas de visualización de alta dimensión para observar de forma rápida los impactos en múltiples dimensiones. Además, el uso de métricas que operen en segmentos específicos puede ser beneficioso.

En cuanto a la deuda técnica por la cascada de correcciones, es común encontrarse con la necesidad de resolver variantes ligeramente diferentes de un problema utilizando modelos de corrección. Sin embargo, esta cascada de modelos crea dependencias y aumenta el costo

del análisis de mejoras futuras. Mejorar un modelo en la cascada puede tener efectos negativos en los modelos posteriores si no se vuelven a entrenar adecuadamente.

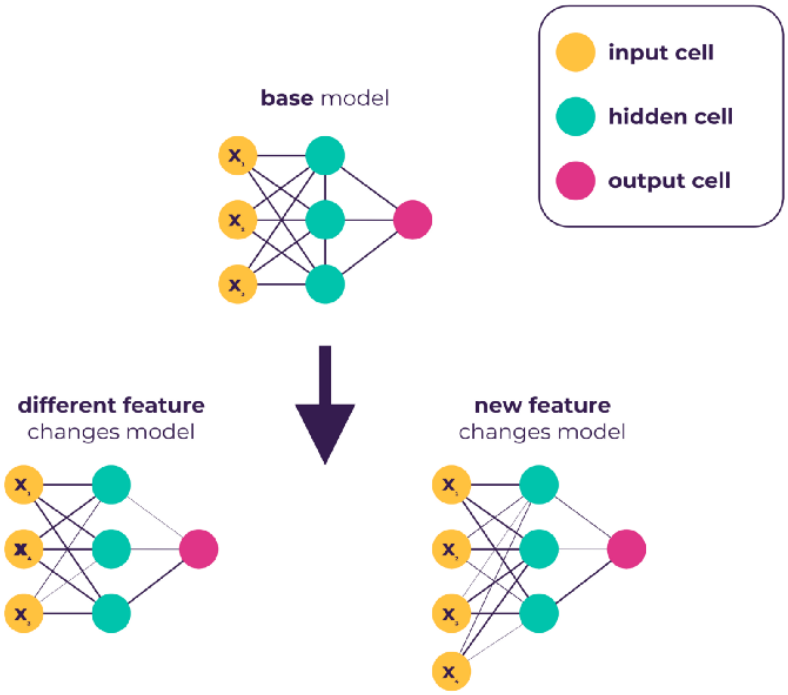



**Figura 2. Añadir una nueva variable de entrada al modelo**

Para mitigar esto, se pueden emplear estrategias como mejorar el modelo original directamente agregando variables adicionales o desarrollar un modelo separado para cada variante del problema como vemos en la *Figura 3*. También se pueden aplicar técnicas avanzadas como la transferencia de aprendizaje, que consiste en utilizar el modelo original como punto de partida para el nuevo modelo y luego entrenarlo con los nuevos datos.

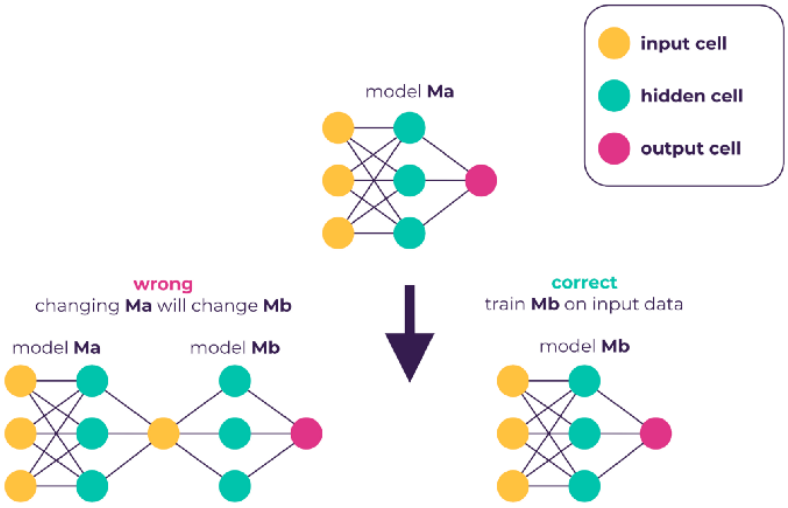



**Figura 3. Cascada de modelos**

Además, es importante considerar los consumidores no declarados, que son los sistemas que utilizan las predicciones de un modelo sin estar debidamente identificados como se muestra en la *Figura 4*. Esto puede ser costoso y potencialmente peligroso, ya que los cambios en el modelo pueden afectar a estos sistemas ocultos y generar bucles de retroalimentación no deseados. Es necesario implementar medidas de protección, como restricciones de acceso y acuerdos de nivel de servicio, para detectar y controlar los consumidores no declarados.

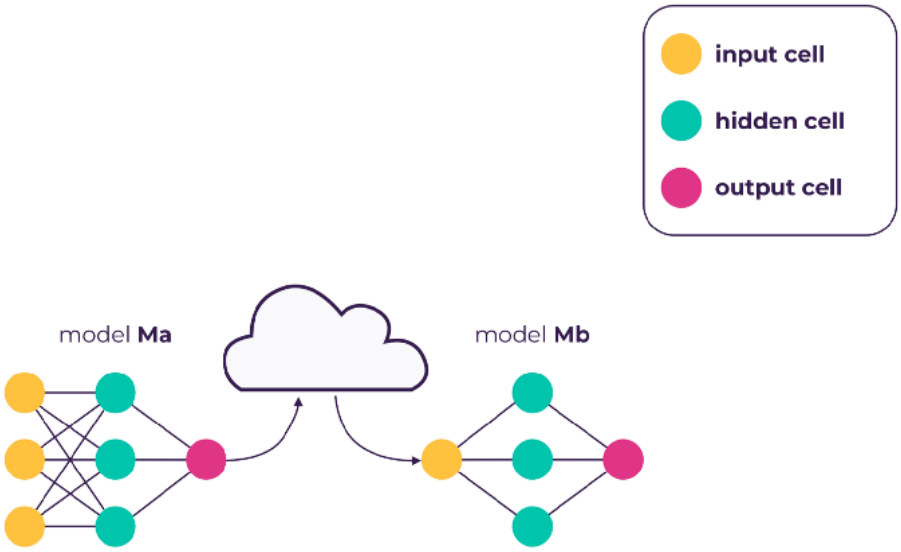



**Figura 4. Consumidores no declarados**

### 2.1.2. Las dependencias de datos

La deuda de dependencia en los sistemas ML, se refiere a las dependencias de datos que pueden resultar complejas de detectar y resolver. A diferencia del código tradicional, donde se pueden detectar dependencias mediante herramientas de análisis estático, en el caso de las dependencias de datos en los sistemas de ML no existen herramientas similares para su detección.

Existen dos tipos de dependencias de datos que pueden generar deuda en los sistemas de ML. En primer lugar, encontramos las dependencias causadas por datos inestables. Esto ocurre cuando se utilizan variables de entrada generadas por otros sistemas, ya que dichas variables pueden cambiar su comportamiento con el tiempo. Por ejemplo, si una variable de entrada proviene de otro modelo de ML que se actualiza periódicamente, las mejoras realizadas en esa variable pueden tener efectos impredecibles en el sistema que la utiliza.

Una estrategia para mitigar este tipo de dependencias es crear copias versionadas de las variables, de forma que se utilice una versión congelada hasta que se haya examinado y validado la nueva versión. Sin embargo, esta solución también implica ciertos costos, como el riesgo de obsolescencia y el mantenimiento de múltiples versiones de la misma variable.

En segundo lugar, se encuentran las dependencias causadas por datos infrautilizados. Estas ocurren cuando se incluyen variables de entrada en el modelo que no aportan muchos beneficios al modelado incremental. Estas variables pueden hacer que el sistema de ML sea vulnerable a cambios innecesarios y potencialmente catastróficos.

Por ejemplo, si se mantienen en el sistema variables de un esquema antiguo de numeración de productos que ya no son necesarias, la eliminación de ese código puede generar problemas. Estas dependencias infrautilizadas pueden surgir de diferentes formas, como la inclusión de variables innecesarias debido a plazos ajustados o la inclusión de variables que no aportan valor real al modelo. Es importante realizar pruebas periódicas para identificar y eliminar estas dependencias infrautilizadas.

En el ámbito del análisis estático de dependencias de datos en los sistemas de ML, no son comunes las herramientas para el análisis estático de código. Sin embargo, existe una herramienta útil para abordar este problema: el sistema automatizado de gestión de variables. Este sistema permite anotar las fuentes de datos y las variables de forma que nos permita realizar comprobaciones automáticas para garantizar que todas las dependencias estén debidamente registradas y que los árboles de dependencia puedan resolverse correctamente.

Estas herramientas pueden facilitar la migración y eliminación segura de dependencias en los sistemas de ML. Aunque no son tan ampliamente utilizadas como las herramientas de análisis estático de código ofrecen una solución eficaz para gestionar las dependencias de datos.

### 2.1.3. Bucles de retroalimentación

En los sistemas de aprendizaje automático en tiempo real, es común que los modelos influyan en su propio comportamiento a medida que se actualizan con el tiempo. Esto puede dar lugar a bucles de retroalimentación que son difíciles de prever y abordar antes del lanzamiento de un modelo. Estos bucles de retroalimentación pueden tomar diferentes formas, pero son especialmente complicados de detectar y solucionar cuando ocurren gradualmente a lo largo del tiempo, especialmente si las actualizaciones del modelo son infrecuentes.

Uno de los tipos de bucles de retroalimentación es el de retroalimentación directa. En este caso, la selección de los datos de entrenamiento futuros puede ser influenciada directamente por el modelo. Aunque se suelen utilizar algoritmos supervisados estándar, una solución teóricamente más adecuada sería utilizar algoritmos *bandit*, que son algoritmos de aprendizaje por refuerzo en los que las salidas del modelo se utilizan como entrada para el mismo. Sin embargo, los algoritmos *bandit* no siempre se adaptan bien a los espacios de acción necesarios para resolver problemas del mundo real, lo que puede generar problemas. Para mitigar estos efectos, se pueden aplicar técnicas como la

aleatorización de datos o el aislamiento de ciertas partes de los datos para evitar que sean influenciados por un modelo específico.

Por otro lado, existen los bucles de retroalimentación ocultos, que son aún más difíciles de abordar. En estos casos, dos sistemas independientes pueden influenciarse indirectamente a través del entorno compartido en el que operan. Por ejemplo, dos sistemas pueden determinar diferentes aspectos de una página web, como los productos a mostrar y las reseñas relacionadas. Mejoras en uno de los sistemas pueden afectar el comportamiento del otro, ya que los usuarios pueden reaccionar de manera diferente a los cambios realizados. Estos bucles ocultos pueden existir incluso entre sistemas completamente separados, como en el caso de dos modelos de predicción del mercado de valores desarrollados por diferentes compañías de inversión. Mejoras o errores en uno de los modelos pueden influir en el comportamiento de oferta y demanda del otro.

Abordar y analizar los bucles de retroalimentación directa representa un desafío estadístico para los investigadores de aprendizaje automático, pero al menos pueden ser estudiados de manera más directa. Sin embargo, los bucles de retroalimentación ocultos son mucho más difíciles de manejar, ya que su influencia es indirecta y puede ocurrir a través de sistemas independientes que interactúan con el entorno compartido. Es importante ser consciente de la posibilidad de estos bucles ocultos y tomar medidas para mitigar sus efectos, como monitorear y analizar de manera continua el comportamiento de los sistemas y realizar ajustes cuando sea necesario.

### 2.1.4. Antipatrones del sistema ML

Los sistemas de aprendizaje automático a menudo contienen una pequeña fracción de código dedicado al aprendizaje o la predicción, mientras que gran parte del código restante se puede describir como "plomería" o infraestructura como se ve en la *Figura 5*. Esto puede generar problemas de diseño de alto endeudamiento en los sistemas de aprendizaje automático, y se han identificado varios patrones de diseño problemáticos que deben evitarse o corregirse.

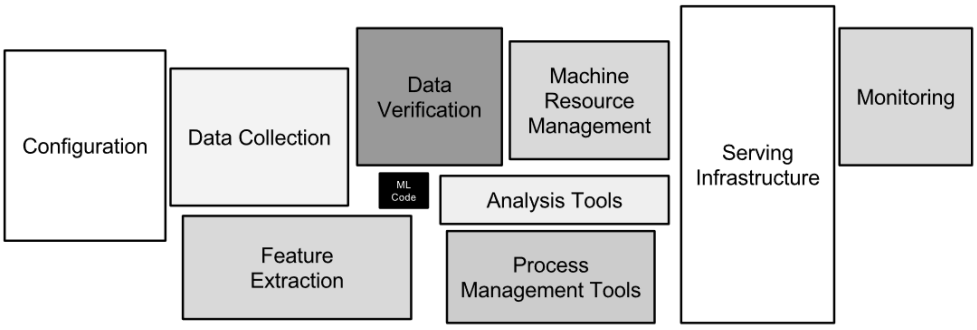



**Figura 5. Composición a nivel de código de un sistema ML**

Uno de estos patrones es el "código pegamento", donde se requiere desarrollar gran cantidad de código para adaptar paquetes de propósito general a casos específicos. Esto puede limitar la flexibilidad y dificultar la mejora o modificación del sistema a largo plazo. Una forma de evitar esto es mediante la creación de API comunes que envuelvan los paquetes de código cerrado, lo que facilita la reutilización y reduce los costos de reemplazo.

Otro patrón problemático es la "jungla de canalizaciones", que suele encontrarse en la preparación de datos. Estas canalizaciones pueden volverse complejas y difíciles de gestionar, lo que afecta la calidad de los datos y al rendimiento del modelo. Se recomienda implementar un monitoreo riguroso en cada una de las etapas de la canalización y distribuir la responsabilidad entre los equipos de ingeniería para asegurar la calidad de los datos.

Los "*codepaths* experimentales muertos" son otra fuente de deuda técnica. Esta práctica implica la creación de ramas condicionales dentro del código de producción para probar métodos alternativos a corto plazo como se ve en la *Figura 6*. Sin embargo, con el tiempo, estas ramas pueden acumular complejidad y dificultar la compatibilidad y las pruebas. Se deben revisar regularmente y eliminar las ramas experimentales que ya no se usen.

La falta de abstracciones sólidas es otro problema en los sistemas de aprendizaje automático. La ausencia de estándares y abstracciones adecuadas dificulta la comprensión e integración de los componentes del sistema, haciendo que sea muy fácil confundir los límites entre los componentes. Es necesario establecer abstracciones sólidas en áreas como el aprendizaje distribuido o la gestión de parámetros.

Por último, algunos de los "*code smells*" más comunes en los sistemas de aprendizaje automático son el uso de tipos de datos antiguos, el uso de múltiples lenguajes de programación y la dependencia excesiva de prototipos. Estos *code smells* pueden indicar problemas subyacentes en el diseño del sistema y se sugiere abordarlos mediante mejores abstracciones e interfaces. En la *Figura 7* se muestra un diagrama con las posibles medidas a tomar cuando detectemos un *code smell*.

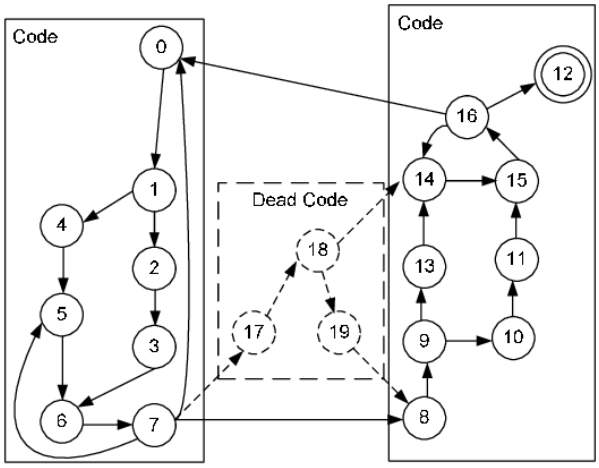



**Figura 6. Codepaths experimentales muertos**

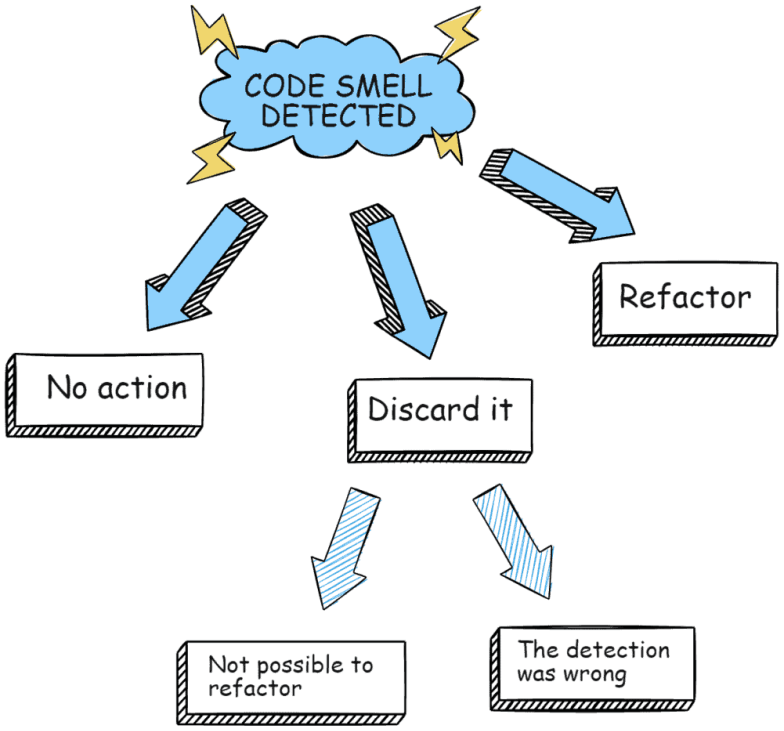



**Figura 7. Como actuar frente a un code smell**

### 2.1.5. Deuda de configuración

La configuración de sistemas de aprendizaje automático puede acumular deuda técnica sin ser detectada. Esto ocurre porque los grandes sistemas tienen numerosas opciones configurables y a menudo se considera la configuración como una tarea tardía, lo que puede llevar a omitir la verificación o prueba de configuraciones aumentando el riesgo de errores potenciales. Algunos ejemplos de problemas comunes son registros incorrectos de funciones, funciones no disponibles en datos anteriores a cierta fecha, cambios en el código debidos a cambios en el formato de registro, funciones no disponibles en producción y la

necesidad de usar funciones sustitutas, asignación de más memoria debido a tablas de búsqueda o restricciones de latencia que pueden impedir el uso de ciertas funciones. La complejidad de la configuración dificulta su modificación y comprensión. Una configuración incorrecta puede generar costosos errores y problemas en la producción. Por este motivo, es importante establecer principios que permitan crear sistemas de configuración efectivos:

- La especificación de una configuración debe ser sencilla y consistir en una pequeña modificación de una configuración previa.
- La tarea de evitar errores manuales u omisiones debe ser sencilla.
- La diferencia de configuración entre dos modelos debe ser visualizable de forma gráfica.
- La verificación automática de los hechos fundamentales sobre la configuración, como el número de variables usadas y las dependencias transitivas de los datos, debe ser sencillo de afirmar.
- Se deben poder identificar configuraciones que no se utilizan o que son redundantes.
- La totalidad de las configuraciones debe ser objeto de revisión exhaustiva y documentarse en un repositorio.

### 2.1.6. Lidiar con cambios en el entorno

Una de las características más atractivas de los sistemas de aprendizaje automático es su capacidad de interactuar directamente con el mundo externo. Sin embargo, el mundo externo es frecuentemente inestable y en constante cambio, lo que implica un costo continuo de mantenimiento.

### 2.1.7. Umbrales fijos en sistemas dinámicos

En el aprendizaje automático, seleccionar un umbral de decisión es crucial para que un modelo realice una tarea específica, como clasificar correos electrónicos como spam o no spam. En el enfoque convencional, los umbrales se eligen manualmente buscando un equilibrio entre métricas como la precisión y el *recall*. Sin embargo, actualizar múltiples umbrales en varios modelos puede ser tedioso y propenso a errores.

Una posible solución es seleccionar los umbrales mediante una evaluación de los datos de validación retenidos. Esto implica utilizar los datos de validación para determinar automáticamente el umbral óptimo en función de las métricas deseadas. Esta estrategia evita la necesidad de ajustar manualmente los umbrales cuando se actualiza el modelo con nuevos datos.

El monitoreo junto con las pruebas unitarias y de sistema son importantes, pero no son suficientes para garantizar la fiabilidad del sistema en entornos cambiantes. Es necesario contar con un monitoreo en vivo del comportamiento del sistema en tiempo real, junto con

respuestas automatizadas para asegurar su funcionamiento a largo plazo. Algunos aspectos que se pueden monitorear incluyen:

- **Sesgo de predicción**: Se puede verificar si la distribución de las etiquetas predichas coincide con la distribución de las etiquetas observadas. Aunque no es una prueba exhaustiva, puede ser útil para diagnosticar problemas y detectar cambios repentinos en el comportamiento del sistema.
- **Límites de acción**: Establecer límites de acción en sistemas que realizan acciones en el mundo real, como pujar por artículos o marcar mensajes como spam, puede servir como un mecanismo de control. Es importante establecer límites lo suficientemente amplios para evitar activaciones por valores anómalos o activar alertas automáticas cuando se alcanza un límite.
- **Productores *Up-Stream***: Si los datos se transmiten a través de múltiples sistemas en un nivel superior, es necesario monitorear y examinar estos procesos para asegurarse de que cumplan con los requisitos del sistema de aprendizaje automático de nivel inferior. Además, es importante transmitir alertas generadas en el nivel superior al sistema de nivel inferior para su control y corrección.

Dado que los cambios externos ocurren en tiempo real, la respuesta a estos cambios también debe ser en tiempo real. Aunque la intervención humana puede ser confiable, es valioso invertir en sistemas que permitan respuestas automatizadas sin depender completamente de la intervención humana.

### 2.1.8. Otras áreas de deuda relacionada con el ML

En el contexto del aprendizaje automático, existen algunas áreas adicionales donde se puede acumular deuda técnica. Estas son:

- **Deuda de prueba de datos**: Si los sistemas de ML utilizan datos en lugar de código, es crucial realizar pruebas exhaustivas en los datos de entrada. Además de las pruebas básicas de coherencia, también son útiles las pruebas más avanzadas que supervisan los cambios en las distribuciones de entrada. Herramientas como *Pandas Profiling*[3] en Python pueden ser utilizadas para analizar y probar los datos de entrada.
- **Deuda de reproducibilidad**: Como investigadores, es esencial tener la capacidad de reproducir experimentos y obtener resultados similares. Sin embargo, lograr una reproducibilidad estricta en sistemas del mundo real puede ser un desafío debido a la aleatoriedad de los algoritmos, el no determinismo asociado al aprendizaje paralelo, la influencia de las condiciones iniciales y las interacciones con el entorno externo.

[3] https://pypi.org/project/pandas-profiling/

- **Gestión de Procesos de Deuda**: En sistemas más complejos y maduros, puede haber múltiples modelos en funcionamiento de forma simultánea. Esto plantea desafíos relacionados con la necesidad de actualizar de manera segura y automática múltiples configuraciones para muchos modelos similares. Además de la gestión y asignación eficiente de recursos entre modelos con diferentes prioridades comerciales junto con la detección y visualización de bloqueos en el flujo de datos en una canalización de producción. También es importante desarrollar herramientas que faciliten la monitorización de incidencias de producción. A nivel de sistema, se debe evitar la presencia de procesos comunes con múltiples pasos manuales, ya que es un claro un indicio de mala práctica de desarrollo, *code smell*.

### 2.1.9. Deuda Cultural

En ocasiones, se establece una distinción clara entre la investigación y la ingeniería de machine learning (ML), pero esta separación puede ser perjudicial para la salud a largo plazo del sistema. Por eso, resulta fundamental fomentar en los equipos una cultura que premie tanto las mejoras en la precisión como la eliminación de variables innecesarias, la reducción de la complejidad, la mejora de la reproducibilidad, la estabilidad o la supervisión. Esto se puede lograr de manera más efectiva en equipos heterogéneos que cuenten con fortalezas en ambos ámbitos, investigación e ingeniería de ML.

#### 2.1.9.1. Conclusiones

La deuda técnica puede ser una buena forma de entender el proceso de desarrollo de software, pero es difícil medir su impacto en el tiempo. Es necesario encontrar una forma de medir su impacto y costo total en un sistema. No dar por sentado que un equipo trabaje correctamente o con baja deuda sólo porque siguen siendo productivos, ya que la deuda técnica sólo se hace evidente después de un cierto tiempo. El hecho de ser capaces de adaptarse de forma ágil frecuentemente añade deuda técnica. Algunas cuestiones a tener en cuenta son:

- ¿Cómo de factible es llevar a cabo pruebas de gran escala para un nuevo enfoque algorítmico?
- ¿Cuál es el nivel de exactitud que se puede lograr al medir el impacto de un cambio reciente en el sistema?
- ¿Es beneficioso mejorar una variable o modelo que degrade a otros en el sistema?
- ¿Cuál es la rapidez con la que se pueden capacitar a los nuevos integrantes del equipo?

## 2.2. Definición MLOps

En la actualidad, uno de los mayores desafíos en el campo del aprendizaje automático es el tiempo transcurrido entre el desarrollo de un modelo y su puesta en producción. Si proceso de puesta en producción requiere mucho tiempo, existe el riesgo de que la vida útil del modelo se vea reducida, ya que el contexto en el que se desarrollo puede haber cambiado y la necesidad a cubrir haber mutado o desaparecido. Es por este motivo, que se debe de acortar el tiempo de puesta en producción para extraer el máximo valor del modelo.

El concepto de MLOps surge por primera vez en el artículo (Hidden Technical Debt in Machine Learning Systems, 2014), aunque todavía no se le conocía como MLOps. El término "MLOps" fue acuñado por Kaz Sato, *Staff Developer Advocate* en Google Cloud. Este fue quien se percató de la posibilidad de aplicar los mismos conceptos de unificar el desarrollo y las operaciones (Dev y Ops), en los sistemas basados en aprendizaje automático.

La metodología MLOps está estrictamente relacionada con el enfoque DevOps. Como su nombre indica, MLOps hereda sus principios de DevOps; sin embargo, la puesta en producción del código de software es diferente a utilizar modelos de aprendizaje automático en producción. Mientras que el código es estático, los datos cambian constantemente. MLOps es un enfoque multidisciplinar en el que un equipo desarrolla proyectos basados en el uso de modelos de aprendizaje automático. Estos proyectos están formados por código, datos y modelos en pequeños incrementos reproducibles que pueden ser ejecutados de manera confiable en cualquier momento, permitiendo ciclos cortos de adaptación.

El principal objetivo del MLOps es reducir el tiempo necesario para poner en producción un modelo. Además, busca proporcionar un enfoque que simplifique y estandarice el ciclo de vida del aprendizaje automático. Para lograr estos objetivos, las organizaciones deben abordar diferentes aspectos técnicos como reducir la brecha entre los científicos de datos y los equipos operativos o adoptar un nuevo enfoque para el desarrollo de modelos de aprendizaje automático.

### 2.2.1. Diferencias entre DevOps y MLOps

DevOps es forma de pensar y trabajar, que optimiza los procesos que aceleran la tasa en la que se obtiene valor empresarial. Tiene sus orígenes en los principios de desarrollo de software ágiles y podría considerarse como una extensión de ellos. DevOps se enfoca especialmente en el primer principio del Manifiesto Ágil: "Individuos e interacciones sobre procesos y herramientas". Al fomentar el pensamiento crítico sobre las herramientas, las considera valiosas, pero no impone ni requiere ninguna en particular. Cabe destacar que, si bien el uso eficaz de herramientas es necesario para lograr una transformación exitosa en DevOps, no es suficiente por sí solo. Esta metodología enfatiza en que las interacciones y

colaboraciones entre individuos son el núcleo del proceso de desarrollo (así como del conjunto de la organización) y que las tecnologías pueden ayudar a mejorarlas.

En un enfoque de DevOps, es importante que los equipos de desarrollo y operaciones compartan información y colaboren estrechamente en la medida de lo posible. Un equipo asume la responsabilidad del subproducto a lo largo de todo su ciclo de vida, evitando transferencias entre desarrolladores y operadores de infraestructura. Además, se hace hincapié en el proceso y no solo en el producto, ya que el enfoque principal recae en el proceso. La automatización desempeña un papel fundamental y se debe aprovechar al máximo para mejorar y simplificar el proceso. Cuando existen tareas repetitivas que pueden ser automatizadas, las personas pueden trabajar de manera más eficiente.

Dentro de DevOps, la automatización desempeña un papel fundamental al permitir la implementación de la Integración Continua (CI) y la Entrega Continua (CD), como se muestra en la *Figura 8*, que representa su evolución. La Integración Continua es el proceso que permite la integración frecuente del nuevo código escrito por los desarrolladores. En contraposición con el enfoque tradicional de trabajo en ramas de características independientes durante semanas, la integración continua busca evitar los problemas de integración que surgen al fusionar grandes fragmentos de código de forma infrecuente. Una vez se realizan los cambios y se fusiona el código, se ejecutan automáticamente los test. Si fallan los test, se identifica que la compilación, build está defectuosa y necesita ser corregida. Este flujo de trabajo permite identificar y resolver problemas rápidamente.

La Entrega Continua (CD) "es la capacidad de obtener cambios de todo tipo, incluyendo nuevas funciones, cambios de configuración, corrección de errores y experimentos, en producción o en manos de los usuarios, de manera segura y rápida de una manera sostenible" (Continuous Delivery)

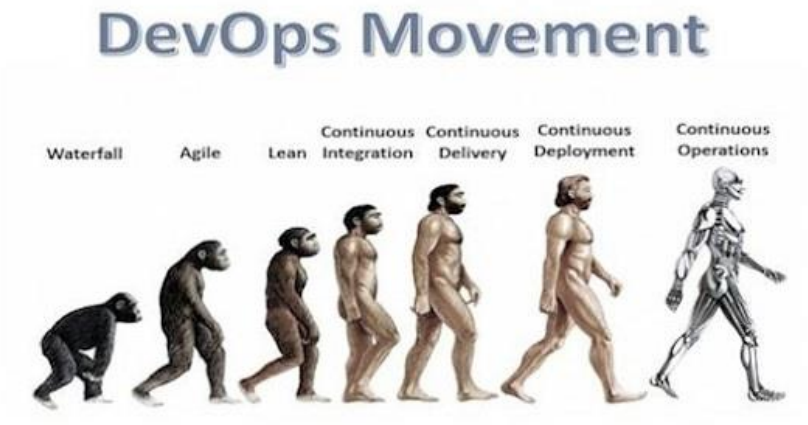



**Figura 8. Evolución del movimiento DevOps**

La metodología MLOps se basa en los principios de DevOps, pero existen algunas diferencias. Ambos buscan reducir el tiempo para poner el proyecto en producción

mediante la simplificación y la estandarización del ciclo de vida. Esta meta se logra mediante la colaboración entre equipos y mejorando la automatización. En DevOps y MLOps, el equipo es responsable del subproducto durante toda su vida útil y no debería haber una transferencia de desarrolladores u operadores de infraestructura. Los equipos deben incorporar a investigadores de aprendizaje automático y científicos de datos, que a menudo no son ingenieros de software experimentados.

Debido a la naturaleza diferente de DevOps y MLOps, también cambia el enfoque de las pruebas, mientras DevOps requiere pruebas para el código, MLOps requiere pruebas también para la validación de datos, validación de modelos y calidad del modelo. Ambos enfoques no imponen el uso de herramientas específicas, pero en ambos casos es importante elegir el instrumento adecuado para poder alcanzar los objetivos.

Otro aspecto importante que diferencia a MLOps de DevOps es cómo se construyen las canalizaciones de Integración Continua/Entrega Continua (CI/CD). En MLOps, los componentes CI deben abarcar la prueba y validación de esquemas de datos, datos y modelos. Mientras que los componentes CD deben permitir el despliegue de la canalización de entrenamiento, así como el servicio o aplicación de predicción del modelo final. Además, hay existe otro componente específico de la metodología MLOps, el de Entrenamiento Continuo (CT), que debe tenerse en cuenta para habilitar el reentrenamiento y la refinación automática del modelo. La *Figura 9* describe el flujo DevOps el cual también es parte del flujo de trabajo de MLOps, pero agregando algunos pasos más relacionados con la gestión de datos y modelos.

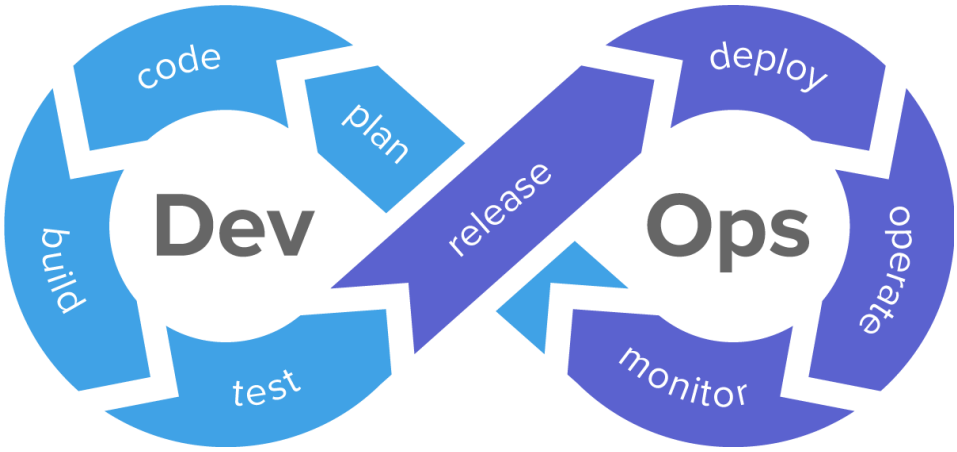



**Figura 9. Flujo DevOps**

### 2.2.2. Otros términos relacionados

Desde el auge del DevOps, han surgido numerosos términos relacionados con las operaciones como: SecOps (para seguridad), NetOps (para redes), ITOps o GitOps. También han surgido otros términos relacionados con los datos, además de MLOps, como DataOps, ModelOps y AIOps. En la *Figura 10* se detallan los ámbitos de cada uno de los términos.

**DataOps**: Este término se utiliza a menudo como sinónimo de MLOps, aunque existen algunas diferencias. “Las principales tareas en DataOps incluyen el etiquetado y pruebas de datos, orquestación de pipelines, control de versiones y monitoreo de datos. Los equipos de análisis y *Big Data* son los principales exponentes de DataOps” (What the Ops are you talking about?). DataOps también se puede relacionar con los analistas de datos, analistas de BI, científicos de datos o ingenieros de datos. El manifiesto de DataOps recuerda al Manifiesto Ágil y se puede resumir como " Individuos e interacciones por encima de procesos y herramientas; análisis de trabajo por encima de documentación exhaustiva; colaboración con el cliente por encima de la negociación de contratos; experimentación, iteración y retroalimentación por encima de un diseño inicial extenso; propiedad multifuncional de las operaciones por encima de responsabilidades aisladas" (DataOps Manifesto).

**ModelOps**: También los términos ModelOps y MLOps a menudo se utilizan indistintamente. ModelOps es más general que MLOps, no solo abarca modelos de aprendizaje automático, sino de cualquier tipo. Su objetivo es el desarrollo de modelos holísticos para el análisis de aplicaciones que permitan realizar un análisis predictivo para mejorar el rendimiento de los procesos.

**AIOps**: AIOps también se puede confundir con MLOps, pero se refiere al proceso de resolver desafíos operativos mediante el uso de la inteligencia artificial. En B*ig Data*, analítica y aprendizaje automático se emplea para recopilar y agregar grandes volúmenes de datos, identificar señales del rendimiento de un sistema sin verse afectado por ruido o para detectar causas raíz de sucesos y resolverlas sin necesidad de intervención humana.

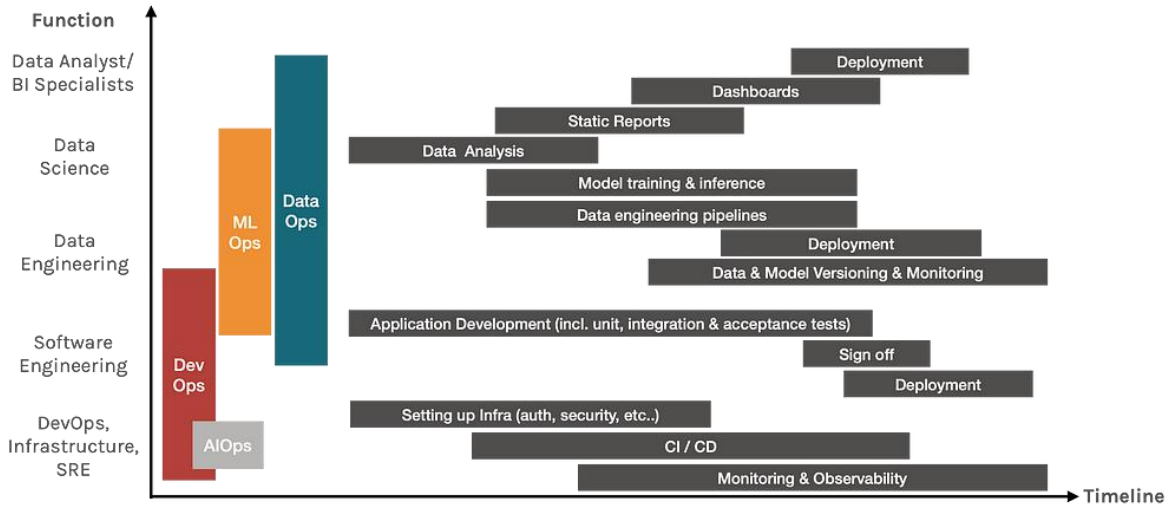



**Figura 10. Tareas cubiertas por cada término**

## 2.3. Ciclo de vida en Machine Learning

DevOps y MLOps tienen en común que ambos ponen en primer plano el proceso antes que el producto. El ciclo de vida del aprendizaje automático no se limita únicamente a la construcción de modelos, ya que la infraestructura de un sistema de aprendizaje automático es amplia y compleja. Solo una pequeña parte de esta infraestructura está compuesta por el código del modelo de aprendizaje automático (representado como el componente negro en la *Figura 11*). Por esta razón, es importante destacar que MLOps se enfoca en todo el ciclo de vida del aprendizaje automático, no solo en la fase de construcción de modelos. En el ámbito de la ciencia de datos, el ciclo de vida del sistema puede variar según el campo específico, pero todos comparten algunos pasos comunes. La *Figura 12* ilustra los pasos del ciclo de vida del aprendizaje automático.

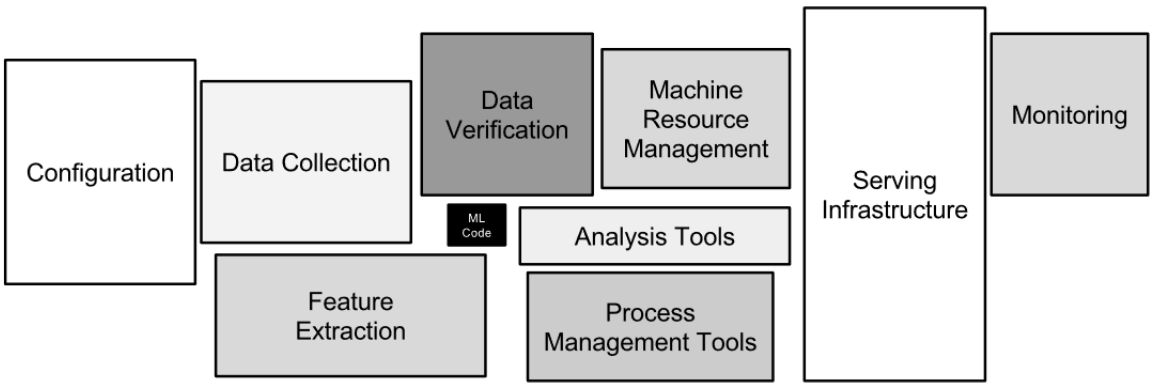



**Figura 11. Ciclo de vida del Aprendizaje Automático**

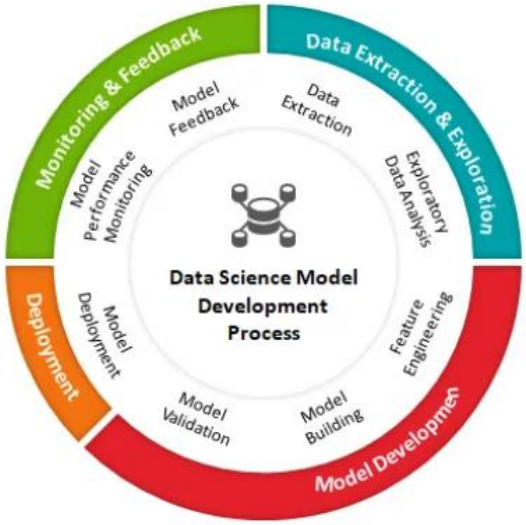



**Figura 12. Pasos del ciclo de vida de aprendizaje automático**

**Extracción y exploración de datos:** En los proyectos de ciencia de datos, la extracción de datos suele ser el primer paso. Los datos pueden estar en diferentes orígenes y formatos. Deben ser extraídos y depurados para poder emplearlos en un análisis más detallado. Tras

la preparación de los datos se pueden explorar en busca de patrones ocultos. La etapa de exploración puede abarcar diversas actividades, como revisar estadísticas, examinar su distribución, buscar correlaciones y llevar a cabo tareas de limpieza, rediseño, filtrado y muestreo de los datos.

**Desarrollo de modelos:** En la fase de desarrollo de modelos, el primer paso consiste en aplicar transformaciones a los datos con el fin de mejorarlos y prepararlos para los algoritmos de aprendizaje automático. Esto incluye la ingeniería de variables, que implica generar variables sintéticas a partir de los datos. El uso de más variables puede resultar en un modelo más preciso, pero también tiene sus desventajas. Por un lado, puede aumentar el costo computacional del modelo, ya que más variables requieren más recursos de cálculo. Además, un mayor número de variables implica una mayor cantidad de entradas, lo que puede complicar la implementación y el mantenimiento del modelo. Un modelo con muchas variables puede volverse menos robusto, ya que es más susceptible a cambios en las variables.

En un enfoque MLOps, es útil automatizar la selección de variables, mediante heurísticas, estimando la importancia de algunas variables para el rendimiento del modelo. Existen servicios y librerías en diferentes lenguajes, como *GAparsimony*[4] en Python, que busca los mejores modelos parsimoniosos de baja complejidad. Esto también favorece la adopción de un *Feature Store*, un conjunto de repositorios de diferentes variables asociadas a entidades comerciales que se crean y almacenan en una ubicación central para su fácil reutilización. Una vez los datos están preparados, el modelo puede ser construido aplicando algoritmos de aprendizaje automático y alimentado con los propios datos. Toda la etapa de desarrollo de modelos incluye evaluar cuán bueno puede llegar a ser un modelo, encontrar los mejores hiperparámetros, equilibrar el subajuste y sobreajuste junto con encontrar un equilibrio entre mejora del modelo y costos de cómputo. Este paso también se refiere a la experimentación, que tiene lugar a lo largo de todo el proceso de desarrollo de modelos, cada decisión importante viene con al menos alguna experimentación. Cuando los modelos están construidos, deben ser validados para asegurar que funcionen como se espera. "Esencialmente, todos los modelos son incorrectos, pero algunos son útiles" - George E.P. Box, estadístico británico del siglo XX (Introducing MLOps, 2020). Es importante evaluar un modelo y comparar su rendimiento con lo que existía antes; esto se puede lograr mediante el uso de métricas, pero no hay una métrica que sirva para todos. En el desarrollo de modelos, pueden repetirse diferentes tareas y la automatización puede simplificar el proceso. Un enfoque MLOps también puede proporcionar herramientas para rastrear hiperparámetros, versionar diferentes modelos, registrar métricas y simplificar las comparaciones de modelos, como veremos más adelante.

[4] https://pypi.org/project/GAparsimony/

**Despliegue:** En esta etapa, los modelos se trasladan desde entornos de desarrollo hasta entornos productivos donde se integran en aplicaciones comerciales. Existen dos tipos principales de despliegue de modelos: Modelo-como-servicio y Modelo Incorporado. En el enfoque de Modelo-como-servicio, el modelo se despliega en un *framework* que proporciona un punto de acceso en forma de API REST para responder a solicitudes en tiempo real. Por otro lado, el enfoque de Modelo Incorporado trata el modelo como una dependencia que se construye y empaqueta dentro de la aplicación consumidora, siendo la opción más sencilla.

En la fase de despliegue, un enfoque de MLOps puede incorporar prácticas de Integración Continua/Entrega Continua (CI/CD) y el uso de contenedores. Esto permite automatizar y agilizar el proceso de implementación de los modelos. Además, en grandes empresas pueden existir plataformas corporativas internas que se adaptan a las necesidades específicas de la organización.

**Monitorización y retroalimentación:** La última etapa es la de monitorización y retroalimentación. Una vez el modelo se encuentra en producción, es necesario monitorearlo para garantizar su correcto funcionamiento y detectar posibles fallos. Esto implica supervisar el rendimiento del modelo, el uso de recursos, la calidad de los datos y la satisfacción del usuario. El monitoreo nos permite identificar cambios en los datos de entrada que podrían indicar errores en los datos o la necesidad de reentrenar el modelo debido a cambios en el contexto en el que fue entrenado.

La retroalimentación también es fundamental, ya que nos permite mejorar continuamente el modelo y adaptarlo a los cambios en el negocio o en los datos. En resumen, MLOps se enfoca en todo el ciclo de vida del aprendizaje automático, desde la extracción y exploración de datos hasta el despliegue y la monitorización en producción. Se basa en procesos automatizados que facilitan la experimentación, la selección de características, la validación y el monitoreo de modelos.

Un enfoque de MLOps también fomenta la colaboración entre equipos de datos, desarrollo y operaciones, con el objetivo de lograr un ciclo de vida de aprendizaje automático más eficiente y efectivo. Además, puede utilizar herramientas y prácticas que mejoran la automatización y la eficiencia en todo el proceso.

### 2.3.1. Un proceso, no solo un producto

La *Figura 12* muestra que cada paso del proceso puede tener subpasos, que podrían considerarse como un "proceso dentro del proceso". Por ejemplo, la fase de "extracción y exploración de datos" consta de dos fases distintas, mientras que el paso de "desarrollo de modelos" puede abarcar la ingeniería de variables, el entrenamiento y la validación de modelos. Del mismo modo, como se muestra en la *Figura 12* la fase de "monitoreo" se compone de dos pasos, monitoreo y retroalimentación. Es importante destacar que la

imagen no es exhaustiva y que el ciclo de vida puede abarcar otras etapas según cada situación específica.

Esta perspectiva del ciclo de vida vista como una sucesión de pasos (y subpasos), permite guiar el proceso y automatizarlo de manera efectiva. Además, podemos tener diferentes canalizaciones, como una canalización de entrenamiento para la fase de entrenamiento y una canalización de predicción para el "despliegue de modelos" y el "monitoreo". El enfoque modular del ciclo de vida también permite a los científicos de datos reutilizar cada componente en diferentes canalizaciones. Por ejemplo, el paso de ingeniería de variables puede ser utilizado tanto en la canalización de desarrollo/entrenamiento de modelos como en la canalización de predicción.

Este enfoque del ciclo de vida, visto como un conjunto de canalizaciones compuestos por componentes reutilizables y modulares, permite tres conceptos principales en la automatización de MLOps: Integración Continua, Despliegue Continuo y Entrenamiento Continuo. Estos conceptos son fundamentales para lograr un ciclo de vida de aprendizaje automático eficiente y efectivo.

### 2.3.2. Roles involucrados durante el ciclo de vida MLOps

Si bien los modelos de aprendizaje automático son principalmente diseñados por científicos de datos, el ciclo de vida completo de un sistema de aprendizaje automático involucra a múltiples roles de diferentes equipos. Una de las principales características de MLOps es promover la colaboración entre equipos. MLOps tiene un impacto en todos aquellos que trabajan en el ciclo de vida del aprendizaje automático y, al mejorar la colaboración, brinda beneficios como una mejor comunicación entre los diferentes equipos.

En el ciclo de vida de un sistema de aprendizaje automático, están involucrados varios roles. Entre los cuales se incluyen a los expertos en la materia, científicos de datos, ingenieros de datos, ingenieros de software y DevOps. Sin embargo, MLOps propone un nuevo rol importante, conocido como ingeniero de MLOps. Su principal objetivo es garantizar la implementación de las prácticas de MLOps.

Los **expertos en la materia** son el primer rol involucrado cuando comienza el ciclo de vida del aprendizaje automático y deben estar comprometidos durante todo el proceso. Los perfiles orientados a datos tienden a carecer de una comprensión profunda del caso de negocio y los problemas que deben abordarse. Un experto en la materia define los objetivos, las necesidades de negocio y los indicadores clave de rendimiento (*KPIs*) que desean lograr. Esta figura tiene un papel, no solo al principio del proceso, sino también al final. A veces, para validar si un modelo está funcionando bien o como se espera, las métricas tradicionales (*precisión*, *recall*, *accuracy*, etc.) no son suficientes y los científicos de datos necesitan la opinión de los expertos en la materia. Por ejemplo, los científicos de datos podrían construir un modelo que tenga una muy alta precisión en un entorno de

producción, pero no proporcione los resultados esperados desde un punto de vista del negocio. Al construir un proceso de MLOps, es importante proporcionar a los expertos en la materia una forma fácil de entender las prestaciones del modelo implementado en términos de negocio. Esto no solo se refiere a las métricas, sino también a los resultados o el impacto del modelo en el proceso del negocio. También sería útil proporcionar una forma de profundizar en las decisiones individuales tomadas por un modelo para entender por qué llegó a esa decisión mediante la interpretación y explicación del modelo.

Los **científicos de datos** a menudo se ven estrictamente involucrados en el paso de construir el modelo en el ciclo de vida del aprendizaje automático, pero en realidad es más amplio; los científicos de datos deben estar involucrados con los expertos en la materia, entendiendo y ayudando a formalizar los problemas de negocio de manera que puedan construir una solución valiosa. Un científico de datos no solo necesita habilidades técnicas, sino que también debe comunicarse efectivamente con otras personas involucradas en el proceso, personas con las que a menudo no lo hacen. Un sistema de MLOps sólido debería ayudar a facilitar y simplificar la colaboración entre científicos de datos y otros perfiles con una infraestructura organizativa adecuada. La recopilación de buenas prácticas MLOps también debería permitir a los científicos de datos tomar acción rápidamente sobre los modelos implementados.

Los **ingenieros de datos** tienen la responsabilidad de diseñar y optimizar la recuperación, almacenamiento, transformación y uso de los datos para alimentar los modelos de aprendizaje automático, lo que significa trabajar estrechamente con los expertos en la materia para identificar los datos correctos y prepararlos para su uso. También trabajan estrechamente con los científicos de datos para resolver cualquier problema de datos que pueda causar que un modelo se comporte de forma indeseable en producción.

Los **ingenieros de software** están involucrados en el desarrollo de software y aplicaciones clásicas, es importante que trabajen conjuntamente con los científicos de datos para garantizar el funcionamiento del sistema completo. Por ejemplo, el código de aprendizaje automático debe encajar en la canalización de CI/CD junto con el resto del software que se está utilizando (por ejemplo, un modelo de aprendizaje automático, construido por científicos de datos, que necesita integrarse con la aplicación o el sitio web utilizado por los últimos usuarios).

**DevOps**, pese a que MLOps nació de los principios de este, pueden coexistir. Los DevOps, dentro del ciclo de vida del aprendizaje automático, son personas que suavizan la transición de desarrollo a operaciones manteniendo la infraestructura. Deben garantizar la seguridad, rendimiento y disponibilidad de los modelos de aprendizaje automático, también son responsables de asegurar que el modelo se implemente en el entorno adecuado y se escala de manera efectiva. El rol de DevOps se vuelve aún más importante cuando se implementa

un sistema de MLOps, ya que deben garantizar que el sistema esté automatizado y sea escalable para adaptarse a las cambiantes necesidades de negocio.

## 2.4. Características de MLOps

La metodología MLOps tiene 8 características principales las cuales son la continuidad, la reproducibilidad, el versionado y registro de experimentos, el testeo, el monitoreo, la modularidad, la automatización y los patrones de diseño de pipelines. Para poder hablar de MLOps se deben de cumplir todas, ellas ya que aportan valor en diferentes ámbitos.

### 2.4.1. Continuidad

Como se ha mencionado en apartado anteriores, la metodología MLOps surge a partir de DevOps. Esta metodología presta especial atención a la optimización de procesos en el ciclo de vida de los proyectos mediante la integración y entregas continuas de código. MLOps también les da gran importancia y añade nuevos conceptos, el entrenamiento continuo y la monitorización continúa, que explicaremos más adelante. Estos 4 conceptos permitirán lograr el objetivo de reducir los tiempos tanto de desarrollo como de puesta en producción de los modelos, incluso tras su puesta en producción.

**Integración continua**: En el software tradicional, la Integración Continua (CI) es la práctica que consiste en probar y validar el código junto con los componentes, además de ejecutar pruebas unitarias cuando hay un cambio en el código fuente. El objetivo de CI es validar de forma ágil que cada nuevo cambio sobre el código fuente sea "bueno" y adecuado para su uso posterior. En MLOps, CI se relaciona con la prueba y validación de datos y modelos. Se busca que cada vez que se añadan nuevas líneas de código al repositorio (que contiene nuestro código de entrenamiento) se desencadene de forma automática la reconstrucción de los elementos que conforman la canalización de Machine Learning: contenedores de entrenamiento, ajuste de hiperparámetros, reentrenamiento del modelo y otros. Dado que en un enfoque MLOps, el ciclo de vida está organizado en una canalización (o más), la Integración Continua se puede lograr ejecutando la canalización de datos y la canalización de entrenamiento cuando hay un cambio en el código.

**Entrega continua**: El concepto de Entrega Continua (CD) se refiere a la capacidad de incorporar cambios de todo tipo, incluyendo nuevas variables, cambios de configuración o corrección de errores de forma ágil y segura en producción. En Machine Learning, la Entrega Continua se trata de entregar una aplicación, basada en código, datos y modelos, en incrementos pequeños y seguros, que puedan ser reproducidos y ejecutados de manera confiable en cualquier instante.

**Entrenamiento continuo**: El Entrenamiento Continuo (CT) no es parte de DevOps, y se enfoca en la actualización automática y el uso de modelos. Cuando los datos utilizados para entrenar los modelos son modificados, ya sea por una actualización o por la incorporación

de nuevos datos, se busca volver a entrenar el modelo por completo mediante la ejecución automática de todo el proceso de entrenamiento. El Entrenamiento Continuo es una herramienta poderosa para enfrentar los cambios en los datos (conocido como "*Data Drift*").

**Monitoreo continuo**: Durante la vida productiva del modelo, este puede cambiar, sufrir alteraciones debidas a cambios en los datos o en los conceptos. El monitoreo se enfoca en la auditoría de los datos de producción y en las métricas del rendimiento del modelo, las cuales están estrechamente relacionadas con las métricas de negocio. Lo que se busca es entender cómo funciona el modelo en producción y activar alertas cuando algo se sale de lo común, y en este último caso, reconstruir el proceso y volver a entrenar el modelo si fuera necesario.

### 2.4.2. Reproducibilidad

La capacidad de reproducibilidad es de gran importancia para MLOps. La reproducibilidad abarca dos dimensiones cruciales para todo proyecto de ML, la reproducibilidad de las ejecuciones y la reproducibilidad de los pasos y procesos a realizar para poder ejecutar. Un sistema que puede ser reproducido favorece la productividad. Sin embargo, alcanzar la reproducibilidad es un desafío mayor en el aprendizaje automático que en el software tradicional, ya que los datos y modelos pueden cambiar con el tiempo y es necesario emplear diferentes mecanismos para poder repetir acciones del pasado en las mismas condiciones. Además, la reproducibilidad es fundamental para la automatización, ya que cuando se pueden reproducir todos los pasos y procesos del sistema, se pueden habilitar y automatizar todos los conceptos relacionados con continuidad mencionada en la anterior característica de MLOps.

### 2.4.3. Versionado y registro de experimentos

En un enfoque de MLOps, es importante mantener versionados tanto los datos como los modelos, ya que es posible que se necesite volver a entrenar un modelo debido a cambios en los datos, en el rendimiento o por el desarrollo de un modelo mejorado. Sin embargo, el versionado en el aprendizaje automático presenta desafíos diferentes a los del código tradicional.

El control de versiones en los datos puede suponer un problema ya que no se pueden emplear sistemas de control de versiones clásicos como *Git*. En ocasiones el volumen de datos es muy elevado y se deben de usar técnicas diferentes como instantáneas incrementales de los datos. En cuanto al modelo, su versionado debe registrar cómo se construye el modelo en sí, incluyendo los parámetros utilizados, el entorno de desarrollo, los artefactos relacionados y las métricas obtenidas durante el entrenamiento.

En el aprendizaje automático, es común realizar numerosos experimentos para obtener buenos resultados en un modelo. El portal web ml-ops.org sugiere el uso de diferentes ramas (G*it*) para cada experimento separado. Algunos argumentan que el concepto de ramas en un repositorio de código difiere de la versión específica que se busca en el aprendizaje automático, lo que podría llevar a una explosión en el número de ramas y dificultar la gestión. Personalmente no comparto este enfoque ya que el concepto de ramas diferente por cada experimento por los problemas comentados.

Para garantizar la reproducibilidad de los experimentos y permitir volver a entrenar un modelo de manera consistente, es importante utilizar prácticas que permitan el versionado y seguimiento adecuados de los experimentos realizados. Esto implica documentar los pasos y parámetros utilizados, así como registrar los datos y las métricas asociadas a cada experimento. Al adoptar estas prácticas, se facilita la comparación entre diferentes modelos y la reproducción de entrenamientos anteriores utilizando los mismos datos y condiciones.

### 2.4.4. Testeo

Al igual que en algunas de las anteriores características MLOps, el testeo de las soluciones de Aprendizaje automático difiere de los testeos a realizar en soluciones de código tradicional. Estas pruebas no solo involucran el código, sino que también incluyen pruebas sobre las características y datos involucrados, el desarrollo del modelo y la infraestructura de ejecución. Por la naturaleza de estos desarrollos hay algunos aspectos que son inherentemente no deterministas, los cuales son complicados de automatizar. Las pruebas automáticas aportan gran valor a las soluciones permitiendo mejorar la calidad general del sistema, las cuales se automatizan haciendo uso de la primera de las características, la continuidad. Las pruebas incluyen:

- Es importante realizar una validación adecuada de los datos y características en un proyecto. Para ello, es necesario verificar que los datos de entrada cumplan con el esquema esperado y los valores sean válidos. Asimismo, se recomienda realizar pruebas unitarias en el código de creación de características para evitar errores. También es importante que los datos cumplan con las políticas correspondientes, como el GDPR, y se realice la verificación programática en los entornos de desarrollo y producción. Por último, es útil realizar pruebas de importancia de variables para evaluar si las nuevas variables tienen un poder predictivo efectivo.
- Para asegurar la calidad del modelo de Machine Learning, es importante realizar pruebas que verifiquen que los algoritmos tomen decisiones en línea con los objetivos de negocio. Es decir, las métricas de pérdida del algoritmo, como el *MSE* o *loss*, deben relacionarse con las métricas de impacto empresarial. De esta manera, se puede asegurar que el modelo es efectivo en la toma de decisiones que generen un impacto positivo en la empresa.

- Es importante validar el sesgo y la equidad del modelo, lo cual implica evaluar tanto su desempeño como la inclusión de datos de entrenamiento equilibrados. Por ejemplo, puede haber una falta de equilibrio en los datos para una variable específica, como género o región, en comparación con la distribución real. Al abordar estos problemas de equidad y sesgo, se puede asegurar que el modelo tome decisiones justas e inclusivas para todos los grupos involucrados.
- Es importante evitar el uso de modelos obsoletos en el sistema de producción, ya que pueden afectar negativamente a la calidad de las predicciones realizadas por el modelo. Para garantizar su eficacia, es necesario asegurarse de que siempre se estén utilizando las versiones más actualizadas y precisas del mismo. De esta manera, se pueden tomar decisiones más precisas y relevantes en tiempo real.
- Al elegir un modelo para un proyecto de Machine Learning, hay que considerar el equilibrio entre el desempeño del modelo y su complejidad. Es decir, es necesario evaluar si los beneficios de utilizar un modelo más sofisticado, como una red neuronal en lugar de un modelo lineal, justifican el aumento en complejidad y costo asociados. Al sopesar estos factores, se puede elegir el modelo adecuado para el proyecto que maximice la precisión y eficacia del mismo sin agregar complejidad innecesaria. También es necesario evaluar el empleo de múltiples variables para el entrenamiento del modelo que puedan dificultar su mantenimiento, además de hacerlo menos robusto frente a cambios en el entorno de los datos. El ideal sería el uso de modelos parsimoniosos que permitan maximizar los resultados minimizando la complejidad del modelo tanto en algoritmo como en características.
- Es fundamental probar la infraestructura utilizada para entrenar y servir los modelos de Machine Learning. Para garantizar la reproducibilidad de los resultados, es necesario que el entrenamiento del modelo con los mismos datos genere resultados idénticos en cada ocasión. Además, es necesario someter la arquitectura a pruebas de estrés para asegurar su estabilidad y eficacia en situaciones de alta carga. También es importante realizar pruebas de integración en toda la canalización de Machine Learning para detectar posibles problemas en el flujo de datos y la comunicación entre los distintos componentes. Finalmente, antes de implementar el modelo, es necesario validar que los resultados obtenidos en el entorno de entrenamiento sean similares a los resultados que se obtienen en el entorno de implementación. De esta manera, se pueden detectar y corregir posibles errores antes de que el modelo se implemente en producción.

### 2.4.5. Monitoreo

En un sistema de Machine Learning, es importante tener en cuenta que el rendimiento del modelo puede degradarse con el tiempo debido a cambios en los datos y a la necesidad de reentrenar el modelo. Una vez el modelo se ha implementado, es importante realizar

auditorías periódicas para asegurarse de que está funcionando correctamente en producción. El monitoreo del modelo implica verificar invariantes de datos y establecer alertas para detectar si los datos de entrada no coinciden con el esquema esperado. También es necesario controlar la estabilidad numérica del modelo y establecer alertas para detectar la aparición de valores no válidos, como *NaNs* o infinitos. Además, es importante monitorear con qué datos se está alimentando al modelo para evitar sesgos de datos de entrenamiento y pruebas. Otro campo de monitoreo es la degradación de la calidad predictiva del modelo sobre los datos servidos. También se debe auditar las acciones de los usuarios para recopilar métricas de recompensa y evaluar si el modelo está teniendo el comportamiento deseado. Por ejemplo, si el sistema muestra recomendaciones de productos, se puede rastrear cuándo el usuario decide comprar el producto sugerido como una medida de éxito. Otros objetos de monitoreo incluyen la antigüedad del modelo en producción y la equidad del modelo, lo que implica analizar las entradas y salidas del modelo en relación con variables como la raza, el género o la edad para evitar sesgos.

### 2.4.6. Modularidad

La modularidad es la características que nos permitirá diseñar arquitecturas de sistema como componentes acoplados de manera flexible. Esta característica es una de las más importantes ya que permite obtener otras características de forma sencilla como son el testeo o la continuidad.

Los equipos pueden probar y desplegar de forma rápida y sencilla los componentes de forma individual, de manera que pueden trabajar diferentes equipos de forma independiente. La clave principal para poder lograr la modularidad es el principio de la contenerización.

La contenerización es una técnica que permite encapsular el código junto con todas sus dependencias de manera que puedan ser ejecutados de forma consistente e independiente a cualquier infraestructura. Es una alternativa a las máquinas virtuales ya que se abstrae del sistema operativo host y puede ejecutarse en cualquier plataforma o nube. La contenerización ha permitido en gran medida el desarrollo de MLOps.

Para lograr la modularidad se puede gestionar el flujo de trabajo como un *pipeline* en el que cada uno de sus pasos lo tengamos contenerizado. Esto permite adaptar de forma sencilla el fujo de trabajo añadiendo, borrando o cambiando cualquiera de los pasos presentes en ese flujo. Una de las ventajas de contenerizar los módulos es poder controlar y conocer en todas las ocasiones las condiciones en que será ejecutado ya que ese contenedor dispondrá del sistema operativo y todas las dependencias necesarias para su ejecución, lo que nos permite reproducir las condiciones de ejecución sin necesidad de pasos extra.

### 2.4.7. Patrón de diseño de canalizaciones de flujo de trabajo

Un Patrón de Diseño es una solución que puede ser adoptada en repetidas ocasiones para resolver problemas recurrentes. El libro "Patrones de diseño de aprendizaje automático" (Machine Learning Design Patterns, 2020) describe MLOps como un patrón de diseño denominado "canalización de flujo de trabajo". Esta característica aborda todas las características de MLOps al centrarse en la creación de una canalización o *pipeline* reproducible de extremo a extremo que contenga y orqueste los pasos del proceso ML. El patrón de diseño de canalización de flujo de trabajo destaca la necesidad de reemplazar las aplicaciones monolíticas por una arquitectura de microservicios, en la que la lógica empresarial se crea y se implementa como paquetes de código aislados. Con los microservicios, una aplicación grande se divide en módulos más pequeños y manejables donde los desarrolladores pueden crear, depurar e implementar de forma independiente.

Ejecutar el código de aprendizaje automático como una canalización es esencial para garantizar que otros puedan reproducir el trabajo de manera confiable. El patrón de diseño "*Workflow Pipeline*" permite a otros ejecutar y supervisar todo el proceso de extremo a extremo tanto en entornos locales como en la nube. Contenerizar cada paso de la canalización asegura que otros puedan reproducir tanto el entorno utilizado para construirlo como el flujo de trabajo capturado en la canalización. Esto también permite un desarrollo más rápido y minimiza los riesgos asociados con un proceso monolítico. Este patrón, utiliza un gráfico acíclico dirigido (*DAG*) y brinda un entorno de flujo de trabajo flexible en el que se puede ejecutar pasos individuales o una tubería completa de extremo a extremo. Además, proporciona registro y monitoreo para cada paso de la canalización a través de diferentes ejecuciones y permite el seguimiento de artefactos de cada uno de los pasos del flujo de trabajo.

### 2.4.8. Automatización

El proceso de Machine Learning se puede acelerar a través de la automatización, ya que permite entrenar nuevos modelos o actualizar los ya existentes de manera más rápida. Por ejemplo, Google ha establecido tres niveles de automatización (Google. MLOps: Continuous Delivery and automation pipelines in Machine) para los sistemas ML. En el primer nivel (nivel 0), todo el proceso se lleva a cabo manualmente. El nivel 1 implica la ejecución automática del entrenamiento del modelo y, finalmente, en la última etapa (nivel 2), se introduce un sistema de CI/CD. No es necesario implementar los tres niveles al mismo tiempo, sino que se pueden realizar gradualmente para mejorar la automatización en el desarrollo y producción del sistema de Machine Learning.

#### 2.4.8.1. Nivel 0 de MLOps: Proceso manual

El nivel 0 se refiere a la construcción y aplicación manual de modelos de Machine Learning, sin la utilización de MLOps. Este nivel es fundamental en muchas empresas que están

comenzando a incorporar el Machine Learning en sus casos de uso y tienen un bajo nivel de madurez. Aunque este enfoque manual puede ser suficiente en situaciones donde los modelos no requieren cambios frecuentes, en la práctica, los modelos a menudo fallan cuando se aplican en el mundo real.

*Características*

En este enfoque, cada paso del proceso es llevado a cabo de manera manual como se puede ver en la *Figura 13*, incluyendo la recolección y análisis de datos, la preparación de datos, el entrenamiento y validación del modelo, junto con la transición entre cada paso. Por norma general es un científico de datos quien controla el proceso ejecutando de forma interactiva código experimental hasta encontrar un modelo que se ajuste a los requerimientos. En este punto vemos que existe una clara desconexión entre los científicos de datos y los ingenieros encargados de su implementación, lo que puede ocasionar sesgos entre el entrenamiento y los resultados en entornos productivos.

Derivada de esta desconexión nos encontramos que las iteraciones de versiones del modelo son poco frecuentes y se espera que el modelo no cambie de forma asidua. No se contempla CI/CD ya que las pruebas de código las realiza de forma manual el científico de datos y el modelo casi no se actualizará. El proceso se centra en la implementación del modelo como un servicio de predicción, en lugar de implementar todo el sistema de ML. En estos modelos no hay supervisión activa del rendimiento del modelo, lo que puede llevar a la degradación del rendimiento y otros desvíos en el comportamiento del modelo.

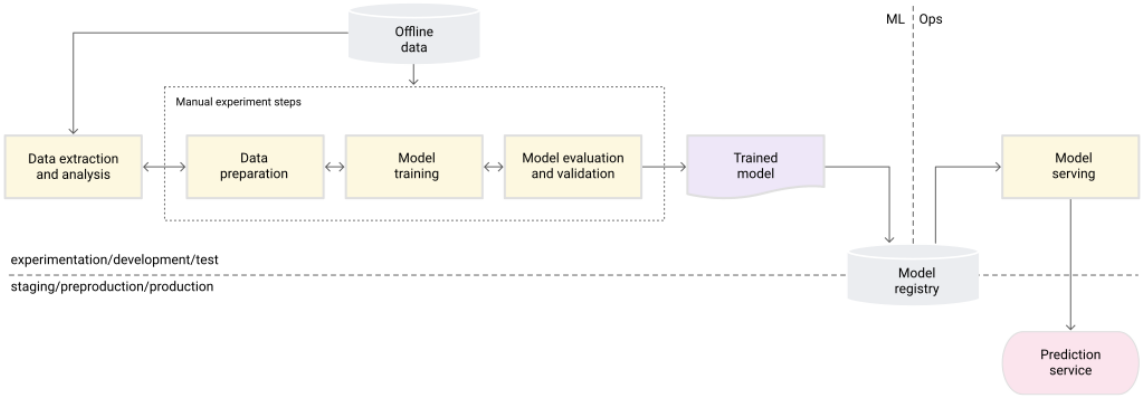



**Figura 13. Nivel 0 MLOps, Proceso manual**

La configuración para las pruebas, la implementación y el ajuste de la API es, pudiendo ser realizado por un equipo de ingeniería. Se realizan varios tipos de pruebas, como pruebas *canary*, de seguridad, regresión y carga. Antes de implementar completamente una nueva versión de un modelo de ML en producción, se suelen llevar a cabo pruebas A/B o experimentos en línea para verificar su funcionamiento.

*Desafíos*

En muchas empresas que comienzan a utilizar el aprendizaje automático en sus casos de uso el nivel 0 de MLOps es bastante común. Aunque este proceso manual, que se basa en datos científicos, puede ser beneficioso cuando se realizan cambios o entrenamientos en los modelos con poca frecuencia, en la práctica, los modelos suelen fallar cuando se implementan en situaciones reales debido a que no se adaptan a los cambios en el entorno y los datos. Para abordar estos desafíos y mantener la precisión del modelo en producción, es importante seguir algunos pasos clave.

En primer lugar, es necesario supervisar activamente la calidad del modelo en producción para detectar la degradación del rendimiento y la inactividad del modelo. Esto puede servir como una señal para realizar una nueva iteración de experimentación y reentrenar manualmente el modelo con nuevos datos. En segundo lugar, se recomienda volver a entrenar con frecuencia los modelos de producción para capturar patrones emergentes y en evolución de los datos más recientes. Por último, es importante experimentar constantemente con nuevas implementaciones para aprovechar las últimas ideas y avances tecnológicos.

Para solucionar los desafíos del proceso manual, se sugieren las prácticas de MLOps para la integración y entrega continua (CI/CD) y los entrenamientos continuos (CT). Al implementar una canalización de entrenamiento de ML, se puede habilitar el CT y configurar un sistema de CI/CD para probar, compilar y proporcionar rápidamente nuevas implementaciones de la canalización de ML.

#### 2.4.8.2. Nivel 1 de MLOps: Automatización de la canalización de ML

El principal objetivo del nivel 1 de MLOps es conseguir el entrenamiento continuo (CT) del modelo mediante la automatización del pipeline de aprendizaje automático, lo que nos permite conseguir la entrega continua (CD). Es necesario incluir los datos automatizados, las etapas de validación de modelos, los activadores de canalización y la administración de metadatos en la canalización para automatizar el proceso de entrenamiento continuo del modelo con datos nuevos en producción. En la *Figura 14* podemos ver un esquema de una canalización de ML automatizada para el CT.

*Características*

Este nivel se caracteriza por experimentos rápidos ya que la transición entre pasos es rápida y automática, permitiendo una ágil iteración entre experimentos y una mejor preparación para industrializar la canalización. Se busca el entrenamiento continuo (CT) realizando los reentrenamientos del modelo en producción con los datos más actualizados. Para poder realizar esto se debe contar con una simetría experimental-operacional entre los entornos productivos y de desarrollo, facilitando de esta manera el cambio de entorno. Este es un aspecto clave del MLOps para ser unificado con DevOps. Además de poder estandarizar los entornos, también se busca estandarizar y modularizar el código, de forma que los

componentes que forman el pipeline sean reutilizables y aislados. Por último, se añaden también la entrega continua (CD) del modelo, cada nueva versión del modelo si cumple unos requisitos y supera unas validaciones es publicado de manera automática en producción.

En el nivel 0, se implementa el modelo entrenado como un servicio de predicción en producción, mientras que en el nivel 1, se implementa una canalización de entrenamiento completa que se ejecuta de forma automática y recurrente para entregar el modelo entrenado como un servicio de predicción.

En este nivel se añaden nuevos componentes:

### Validación de datos y modelos

Cuando se automatiza un pipeline capaz de reentrenar modelos de forma automática es crucial el poder asegurar que tanto los datos con los que se ha generado este nuevo modelo como el modelo resultante con correctos.

### Feature Store

Es una plataforma que centraliza la definición, almacenamiento y acceso de variables para su uso en el entrenamiento y servicio del modelo, asegurando que estas estén estandarizadas. Su principal objetivo es evitar la necesidad de recrear variables que ya estén disponibles, fomentando así la reutilización.

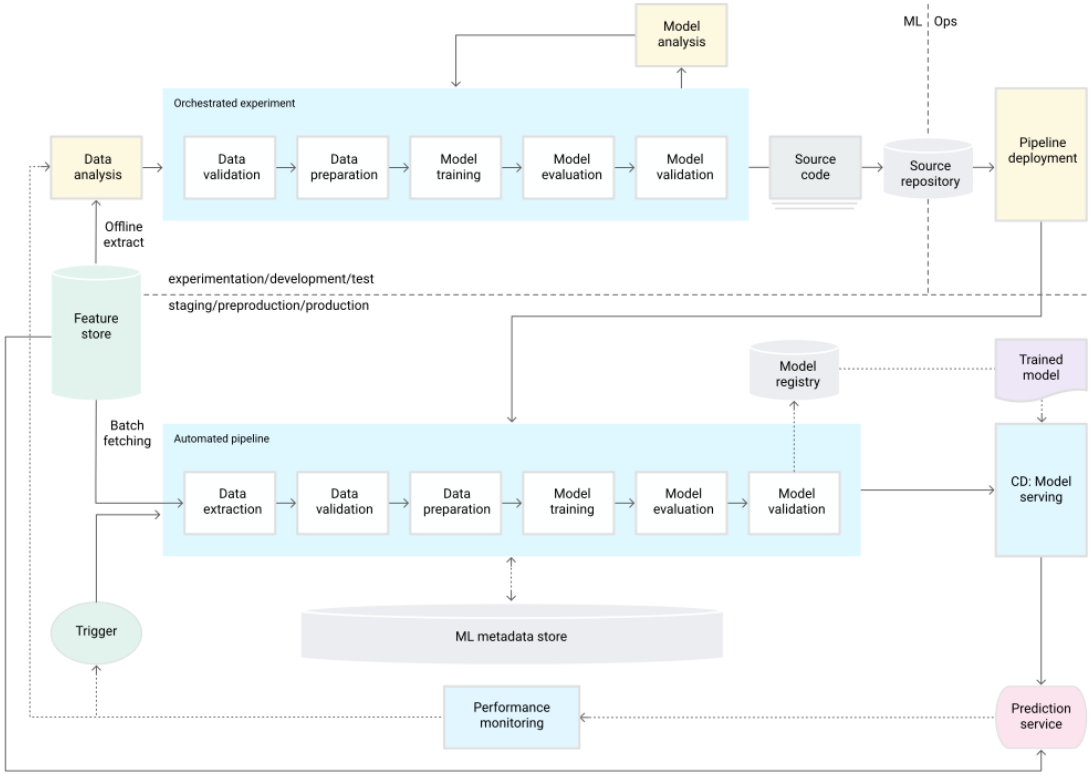



**Figura 14. Nivel 1 MLOps, automatización de la canalización de AA**

Gestión de los metadatos

Es importante tomar nota de la información de cada ejecución de la canalización con el fin de mejorar la reproducibilidad y hacer comparaciones. Cada vez que se ejecute la canalización, se pueden registrar diferentes metadatos, como la versión de la canalización y los componentes utilizados, el tiempo de inicio y finalización (y su duración), los parámetros utilizados y las métricas obtenidas durante la evaluación del modelo.

Pipeline triggers

Existen diferentes situaciones que pueden desencadenar la ejecución del *pipeline* de forma automática, como la solicitud manual, la programación de una ejecución regular (ya sea diario, mensual u horario), la disponibilidad de nuevos datos de entrenamiento, la detección de una disminución en el rendimiento o bien, la aparición de cambios notables en la distribución de los datos.

*Desafíos*

Cuando se trata de un número limitado de canalizaciones que no requieren actualizaciones frecuentes, es común realizar evaluaciones manuales de la canalización y sus componentes, y llevar a cabo nuevas implementaciones de forma manual. Este sistema de trabajo donde un equipo es el encargado de implementar el código en el entorno objetivo es adecuado para modelos nuevos.

Si lo que se busca es probar ideas o realizar implementaciones nuevas de los componentes, contando con muchos pipelines en producción, lo correcto es contar con una configuración de CI/CD que nos permita automatizar todo el flujo de compilación, prueba e implementación.

#### 2.4.8.3. Nivel 2 de MLOps: Automatización de la canalización de CI/CD

Para obtener actualizaciones confiables y rápidas de las canalizaciones en producción, es esencial contar con un sistema de CI/CD automatizado y sólido. Esto permite a los científicos de datos explorar rápidamente nuevas ideas relacionadas con la ingeniería de variables, arquitectura de modelos e hiperparámetros. Estas ideas pueden ser aplicadas en los nuevos componentes de la canalización para ser compilados, evaluados e implementados automáticamente en el entorno de destino mediante el sistema de CI/CD automatizado. En la *Figura 15* se muestra un pipeline de ML con CI/CD, junto con los procesos de ML y las rutinas de automatización de CI/CD.

Este nivel de MLOps cuanta con los siguientes componentes:

- Control de la fuente
- Servicios de compilación y prueba
- Servicios de implementación
- Registro de modelos
- Feature Store

- Metadata Store
- Organizador de canalizaciones ML

*Características*

En la *Figura 16* se detalla el flujo con las etapas de un pipeline de automatización CI/CD. El pipeline está formado por las siguientes etapas:

1. Desarrollo y experimentación: Esta fase se centra en la creación y pruebas de nuevas soluciones de aprendizaje automático. Se llevan a cabo investigaciones para descubrir nuevos algoritmos y construir modelos. Al final de esta etapa, se obtiene el código fuente de los diferentes pasos de la canalización, el cual, se guarda en un repositorio de código fuente.
2. Pipeline de Integración Continua (CI): En esta fase se procede a la compilación del código fuente y la realización de diversas pruebas. Los resultados obtenidos en esta etapa serán los componentes de la canalización, como paquetes, ejecutables y artefactos, que se utilizarán en la implementación posterior.
3. Pipeline de Entrega Continua (CD): Esta fase es la que se encarga de implementar los artefactos generados en la etapa de CI en el entorno de producción. Como resultado de esta fase, obtendremos un pipeline completamente implementado con la nueva versión del modelo.
4. Entrenamiento Continuo (CT): Se puede programar la ejecución automática de la canalización en producción o activarla en respuesta a algún evento específico (trigger). Tras esta etapa, se obtiene un modelo entrenado que se almacena en el registro de modelos.
5. Entrega Continua de modelos: El modelo entrenado es entregado como un servicio de predicción que se utiliza para realizar predicciones. La salida de esta fase es un servicio de predicción del modelo que ha sido implementado.
6. Monitoreo: Se obtienen estadísticas acerca del rendimiento del modelo en relación a los datos que están siendo procesados. El resultado de esta fase es una señal que activa la ejecución de la canalización o la realización de un nuevo ciclo experimental.

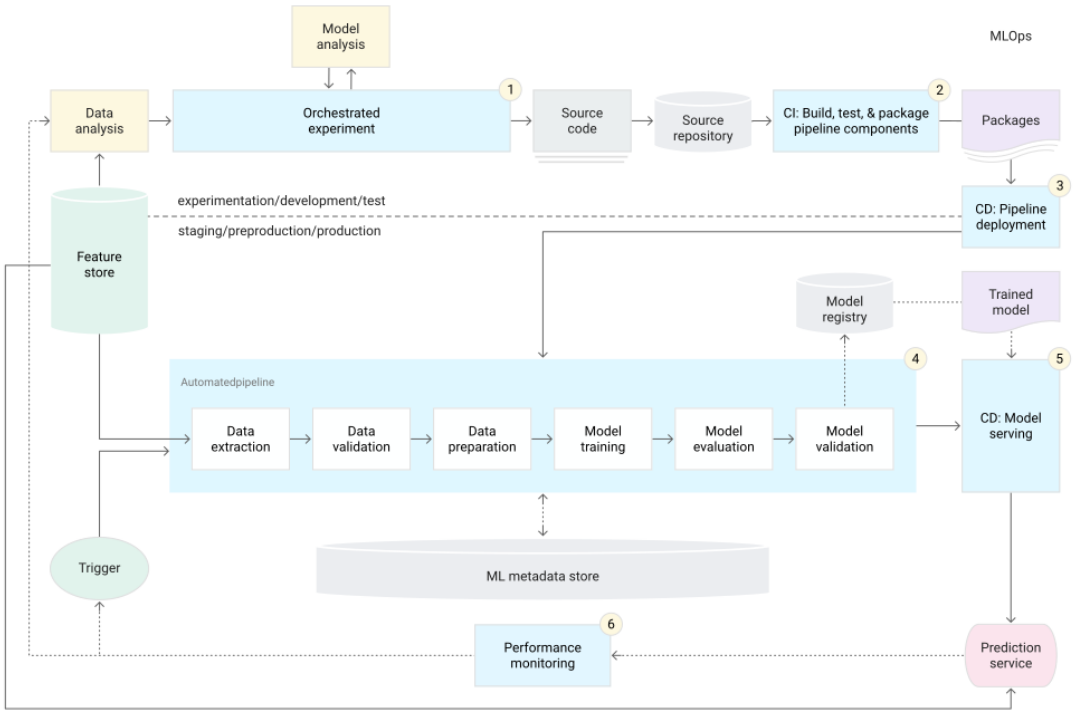



**Figura 15. Nivel 2 de MLOps, automatización de la canalización de CI/CD**

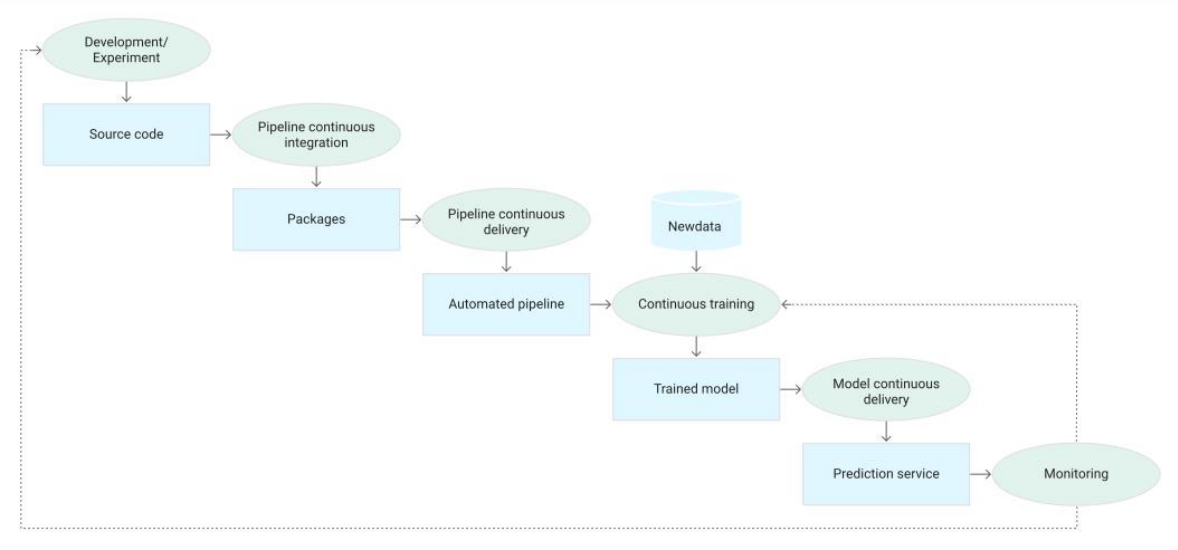



**Figura 16. Etapas pipeline ML de automatización CI/CD**

## 2.5. Retos

En un sistema de aprendizaje automático, el verdadero desafío no es construir un modelo, sino construir un sistema integrado y operarlo continuamente en producción, es decir, manejar el sistema como un proceso en lugar de como un producto. Desde este punto de vista, se busca automatizar tanto como sea posible todo el ciclo de vida y en la medid de lo posible, ser capaces de reproducir todos los pasos del proceso de manera sencilla. Otros objetivos al desarrollar y construir un sistema de aprendizaje automático incluyen:

- Aumentar la colaboración entre diferentes equipos.
- Elegir las herramientas adecuadas de entre una amplia variedad de *frameworks* y herramientas diferentes.
- Validar los datos, para afrontar el problema previamente mencionado sobre la falta de validación de datos.
- Registrar cómo crear el modelo de forma sencilla y repetitiva en cualquier momento. También es importante hacer un seguimiento transparente de los parámetros usados durante el entrenamiento del modelo y sus métricas.
- Lidiar con habilidades heterogéneas entre los individuos que participan en el ciclo de vida de Machine Learning es importante, ya que las personas de los equipos de negocios, ciencia de datos y TI pueden tener herramientas y habilidades diferentes o incluso no compartir habilidades en algunos casos. (Introducing MLOps, 2020).
- Es importante realizar pruebas del modelo, y es necesario destacar que las pruebas unitarias del desarrollo de software tradicional no son suficientes para evaluar el modelo de manera adecuada.
- Es necesario hacer frente a muchas dependencias, ya que no solo los datos están en constante cambio, sino que las necesidades empresariales también cambian con frecuencia. (Introducing MLOps, 2020).
- Es importante enfrentar el problema de la deriva en el aprendizaje automático, ya que los modelos pueden fallar al adaptarse a los cambios en la dinámica del entorno o en los datos utilizados. Evitar la deriva de datos es un desafío complejo, pero necesario para garantizar la precisión del modelo. Como dijo el filósofo griego Heraclitus, "el cambio es la única constante en la vida".
- El proceso de mover un modelo de preparación a producción se realiza automáticamente sin intervención humana.

# 3. Contexto actual de MLOps

MLOps no requiere el uso de un conjunto específico de herramientas, pero es importante elegir algunas que permitan construir una buena infraestructura que satisfaga los requisitos empresariales. Existe un gran abanico de herramientas diferentes que permiten abordar los desafíos de MLOps mencionados anteriormente. A continuación, se comentarán algunas de las herramientas de código abierto más utilizadas, las cuales a menudo se encuentran integradas en soluciones comerciales. Los principales proveedores de la nube, como Google (Google. MLOps: Continuous Delivery and automation pipelines in Machine) (Setting up an MLOps environment on Google Cloud), Amazon (Aws MLOps Framework) y Microsoft (Azure MLOps Framework) cuentan con herramientas para MLOps. Las herramientas de código abierto suelen proporcionar soluciones para problemas específicos de MLOps, mientras que las herramientas de los proveedores de la nube ofrecen entornos completos para trabajar con MLOps. Por ejemplo, Databricks y Google integran herramientas de código abierto como *MLFlow* y *Airflow* en sus soluciones comerciales.

## 3.1. Necesidad actual

MLOps es un enfoque nuevo que se encuentra en constante evolución. Debido a su juventud, todavía no existe un manifiesto claro con una definición compartida y certificada. Una razón más interesante para la falta de un manifiesto común proviene de la comunidad DevOps y también podría aplicarse a MLOps: "sería el fin de la discusión. Y ese es exactamente el problema con un manifiesto. La comunidad le gusta mejorar, educar, animar, informar y energizar a las personas. Básicamente, todos están abiertos a abrazar cualquier cosa que ayude" (Why Is There No DevOps Manifesto?).

MLOps también es muy flexible en cuanto a tecnologías. Hoy en día, los proveedores *Cloud*, como Google, Amazon, Microsoft o Databricks, ofrecen soluciones para adoptar MLOps. También existen una gran cantidad de potentes herramientas de código abierto con sus respectivas comunidades las cuales están muy activas y comprometidas en mejorar sus herramientas. Muchas empresas están comenzando a usar un enfoque basado en MLOps para desarrollar un sistema de aprendizaje automático. A través de una búsqueda en Google Trends, como se ve en la *Figura 17*, se puede apreciar que el tema de MLOps está subiendo en interés. Google Trends pone a MLOps como una de las tendencias que más promete aumentar. Las soluciones MLOps se volverán cada vez más necesarias a medida que la mayoría del mercado adopte la IA. El mercado de MLOps es relativamente inmaduro e incipiente, con soluciones tecnológicas que surgen continuamente. De hecho, se ha predicho que será una de las tendencias más importantes en el futuro.


**Figura 17. Interés en Google Trends a lo largo del tiempo del tema: MLOps**

Por otra parte, el 28% de los proyectos de IA/aprendizaje automático fallan debido a la falta de experticia necesaria, datos listos para la producción y entornos de desarrollo integrados. Muchos más proyectos (47%) fallan en la fase de experimental y en producción. La comunidad Kubeflow[5] realizó una encuesta en marzo de 2021 sobre los beneficios, brechas y requisitos para el aprendizaje automático, también describiendo la necesidad de MLOps. Según la encuesta, la mayoría de los modelos de aprendizaje automático tienen una vida relativamente corta: el 50% se ejecuta en producción durante 3 meses o menos. Por otro lado, el 25% de los modelos permanecen en producción durante 6 meses o más. En la *Figura 18* podemos apreciar una estimación de tiempos en producción de los modelos.

MLOps entra en juego para hacer frente a diferentes "deudas técnicas", anti-patrones y desafíos comunes en el desarrollo y despliegue de un sistema de aprendizaje automático.

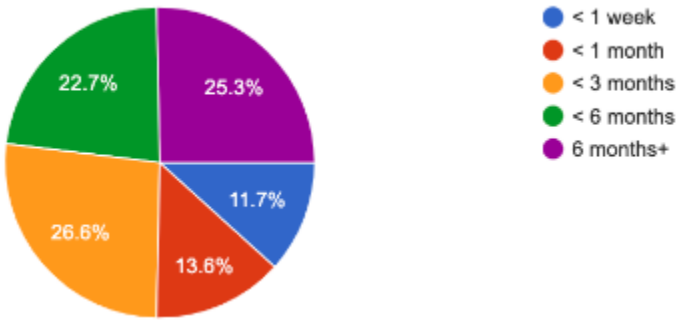



**Figura 18. Duración en producción de los modelos**

## 3.2. Tecnologías actuales para MLOps

El uso de software de código abierto en el ámbito empresarial implica hacer un balance entre la fiabilidad y perspectivas de la herramienta, por ejemplo, la posibilidad de mejoras continuas a través del trabajo de una gran comunidad que soporte y mantenga la herramienta. A pesar de que una comunidad sólida es una ventaja para el crecimiento y el soporte del usuario, uno de los principales inconvenientes del software de código abierto es la falta de calidad en la documentación, aunque no tiene por qué ser siempre así.

[5] https://www.kubeflow.org/

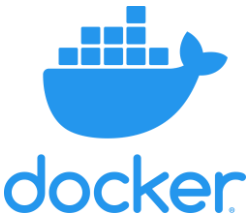
### 3.2.1. Entorno y Contenerización

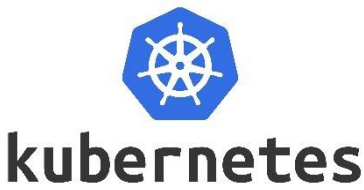
**Docker** es una plataforma de código abierto que utiliza la idea de la contenerización para desarrollar y ejecutar aplicaciones en un entorno aislado. Esta herramienta permite la separación de las aplicaciones de la infraestructura, junto con la reducción del tiempo necesario para ejecutar una aplicación tras el desarrollo de su código. Docker es portable y ligero, lo que facilita la gestión dinámica de cargas de trabajo, el escalado de aplicaciones y servicios en tiempo real.

En una infraestructura de MLOps, Docker permite empaquetar y aislar los pasos de una canalización en microservicios para reproducirlos en diferentes entornos. En combinación con Kubernetes y otras herramientas, permite construir una canalización de Aprendizaje Automático CI/CD.

**Kubernetes** es una herramienta de código abierto que permite gestionar aplicaciones que constan de múltiples contenedores (Docker). Su función es mantener la coherencia en todas las fases de desarrollo y producción, permitiendo la adopción de un enfoque basado en microservicios manejando sistemas distribuidos y modulares. Con esta herramienta conseguimos gestionar entornos en los que disponemos de alta disponibilidad y balanceo de carga entre otros beneficios. Además, se sigue el patrón de diseño de propio de Kubernetes, en el que las definiciones de infraestructura son declarativas y cada versión de un recurso se compromete con el control de origen.

En MLOps, Kubernetes se utiliza ampliamente junto con Docker para permitir la orquestación y el CI/CD de Machine Learning. Se utilizan diferentes herramientas, como Kubeflow para la orquestación de la canalización, Jenkins y JenkinsX para el CI/CD, y Seldon Core y KServe para servir el modelo en producción.




### 3.2.2. Registro de Experimentos

Tanto la ciencia de datos como el aprendizaje automático requieren una gran parte de experimentación, la cual se basa procesos iterativos de los cuales se almacenarán métricas y otros datos que permiten comparar todas las pruebas realizadas y con los datos asociados a esa prueba de forma reproducible.

**MLFlow** es una herramienta de código abierto que permite integrar los principios de MLOps en un proyecto de aprendizaje automático con cambios mínimos en el código existente. Incluyendo unas pocas líneas de código se puede realizar el seguimiento de todos los

detalles relevantes de las pruebas realizadas en el proyecto. Podemos guardar el modelo para uso futuro en implementación junto con todas las métricas que permiten realizar comparativas entre modelos individuales, lo que ayuda a seleccionar el mejor modelo. También podemos registrar el entorno en el que se realiza un experimento. Esta herramienta proporciona una forma de empaquetar el código de aprendizaje automático de forma reutilizable y reproducible, lo que permite compartir el código (y su entorno) con otros científicos de datos, mejorando la colaboración. Otra característica de MLFlow es que proporciona un Modelo Central para almacenar y gestionar los modelos de manera colaborativa. A continuación, se enumeran las principales características de MLFlow:

- MLFlow Tracking: es un componente de MLFlow que permite la reproducibilidad, automatización, comparabilidad y filtrado de experimentos según diferentes criterios. En las *Imágenes 1* y *2* se muestran dos ejemplos de parámetros, métricas y otra información registrada por MLFlow.
- MLFlow Project: es una función permite hacer un seguimiento del entorno y las dependencias necesarias para ejecutar un experimento. Se registra en un archivo llamado MLProject con el entorno conda[6] y el punto de entrada de la ejecución.
- MLFlow Model Registry: es una característica muy útil de MLFlow que permite gestionar y versionar de forma colaborativa los modelos de machine learning. Como el modelo puede cambiar con el tiempo y necesitar ser reentrenado, es necesario gestionar varias versiones del modelo. El Model Registry es una especie de tienda centralizada de modelos que almacena información sobre el entorno en el que se ha entrenado el modelo, sus dependencias, los artefactos producidos y el propio modelo. Cada modelo registrado tiene un nombre único, contiene varias versiones y puede tener diferentes etapas de desarrollo. Cada versión de un modelo puede asignarse a una etapa de desarrollo en cualquier momento, y se puede cambiar de manera programática utilizando la biblioteca de MLFlow. Las etapas predeterminadas son "*Staging*", "Producción" y "Archivado".
- MLFlow Models es un formato estándar para empaquetar modelos de Machine Learning que se pueden utilizar de dos formas diferentes: servicio en tiempo real, a través de una API REST, o inferencia por lotes. Los modelos de Machine Learning se pueden guardar en diferentes “sabores”[7].

[6] Conda es un sistema de administración de paquetes y un sistema de administración de entornos de código abierto que se ejecuta en Windows, macOS y Linux. Conda instala, ejecuta y actualiza rápidamente los paquetes y sus dependencias. (https://docs.conda.io/en/latest/)

[7] Los sabores son una convención que las herramientas de implementación pueden utilizar para entender el modelo; esto hace posible escribir herramientas que funcionen con modelos de cualquier biblioteca de ML sin tener que integrar cada herramienta con cada biblioteca. MLFlow define varios sabores estándar, por ejemplo, para modelos desarrollados a través de Scikit-learn, Tensorflow, Keras, XGBoost, Spark (u otras bibliotecas).

- MLFlow Model Serving: el proceso de poner un modelo en producción o "*serving*", es una tarea que suele ser complicada debido a que el entorno en el que se construyó el modelo suele ser distinto al del entorno de producción. Para abordar este problema, MLFlow proporciona herramientas que permiten ejecutar modelos en un entorno local y exportarlos a contenedores Docker o plataformas comerciales de producción como Azure ML o Amazon SageMaker.

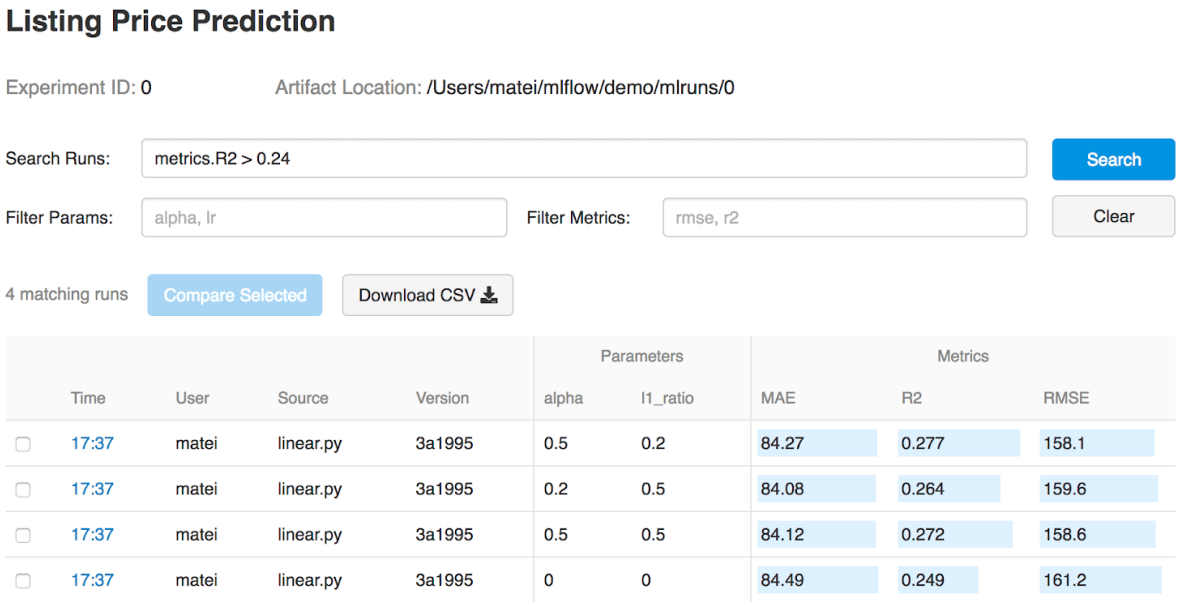



**Imagen 1. MLFlow, registro de experimentos**

**Neptune** es una herramienta que permite, al igual que MLFlow, gestionar experimentos registrando todos los metadatos generados durante los experimentos. Neptune se encarga de almacenar los metadatos generados durante el proceso de MLOps, funcionando como un enlace entre las diferentes etapas del ciclo de vida del modelo de ML, desde la versión de datos hasta el registro y monitoreo de modelos. Su función principal es facilitar la visualización, organización y comparación de estos metadatos de forma intuitiva y limpia. Con esta herramienta, es posible registrar diferentes tipos de datos, como métricas, hiperparámetros, visualizaciones, videos y código, así como organizarlos según una estructura personalizada. Además, permite crear *dashboards* personalizados para compartir con compañeros, gerentes y partes interesadas externas. Las principales diferencias que encontramos con MLFlow son:

- Es de pago.
- Permite gestión de roles y usuarios.
- Es escalable.
- Permite integraciones con *Jupyter notebooks*, *PyTorch*, *Kendro*, etc.

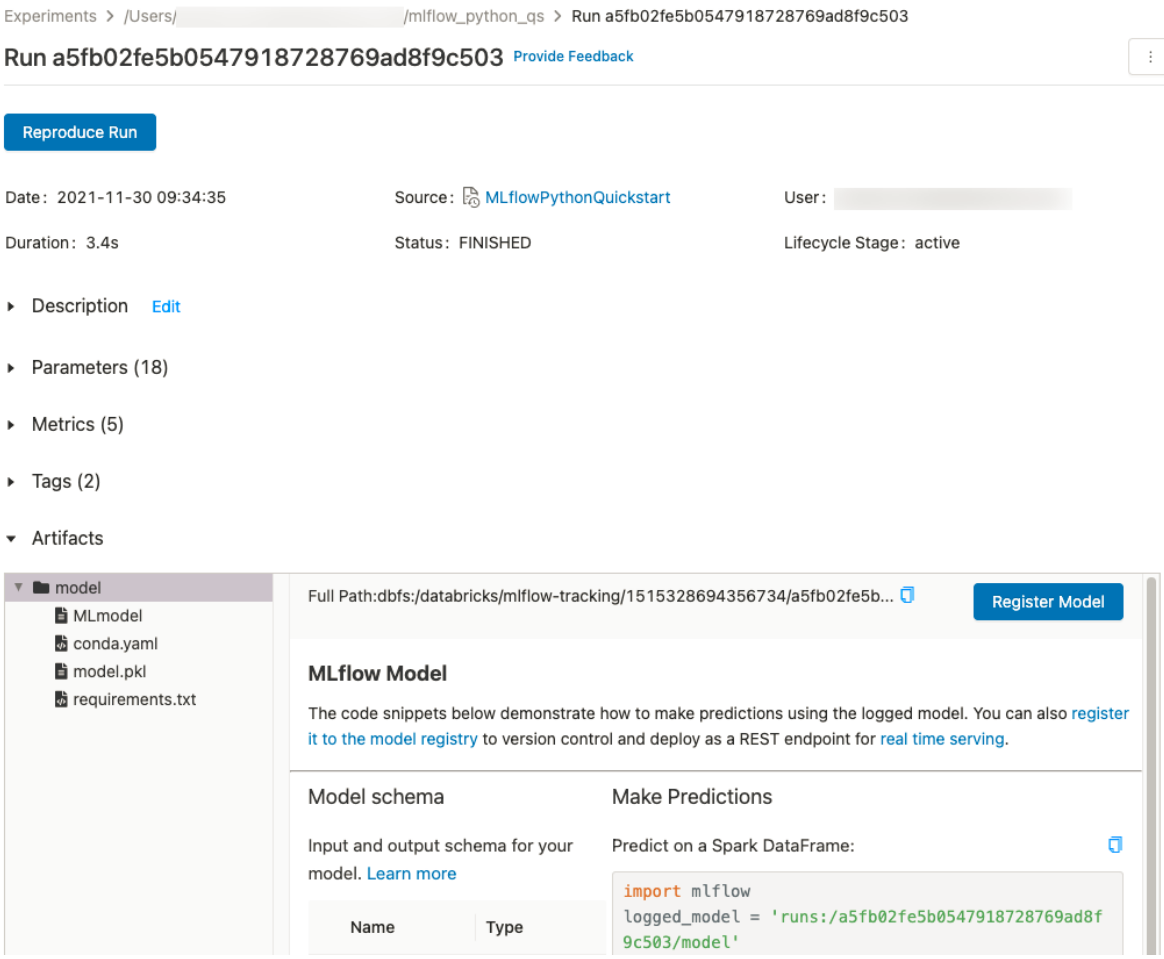



**Imagen 2. MLFLow, detalle de un experimento**

### 3.2.3. Orquestación de Pipelines

**Kubeflow** es un proyecto de Google enfocado en simplificar, hacer portable y escalable la implementación de flujos de trabajo de Machine Learning en Kubernetes. Esta herramienta se basa en Kubernetes, lo que significa que puede ejecutarse en cualquier lugar donde se disponga de esta herramienta. El objetivo de Kubeflow es hacer que la escalabilidad de los modelos de Machine Learning y su implementación en producción sea lo más sencilla posible, aprovechando las ventajas de Kubernetes (facilidad, flexibilidad, repetibilidad, portabilidad para implementaciones en diferentes infraestructuras; implementando y gestionando microservicios con baja interdependencia y escalando según la demanda). Kubeflow tiene varios componentes, como Notebooks para crear y administrar Jupyter, KServe para implementar modelos de Machine Learning en Kubernetes, Katib[8] para la optimización de hiperparámetros y muchos otros. En la *Figura 19* se muestra el uso aproximado de las diferentes herramientas. El componente principal y más utilizado de esta

[8] Es una librería Python para el ajuste de hiperparámetros. https://github.com/kubeflow/katib

herramienta es "Kubeflow Pipelines". Los pipelines de Kubeflow permiten modelar el flujo de trabajo de Machine Learning como una secuencia de pasos, donde cada uno puede recibir datos de entrada y producir uno o más resultados. Un grafo acíclico dirigido (DAG) define la secuencia ordenada de pasos y las dependencias entre los diferentes componentes de la tubería. Cada tarea en el DAG se puede ver en la interfaz de usuario de la herramienta. Kubeflow permite realizar un seguimiento de todos los experimentos, visualizando cada ejecución y los parámetros utilizados (ya sea que hayan tenido éxito o no). Durante cada ejecución, se registran diferentes datos, como métricas o registros de ejecución.

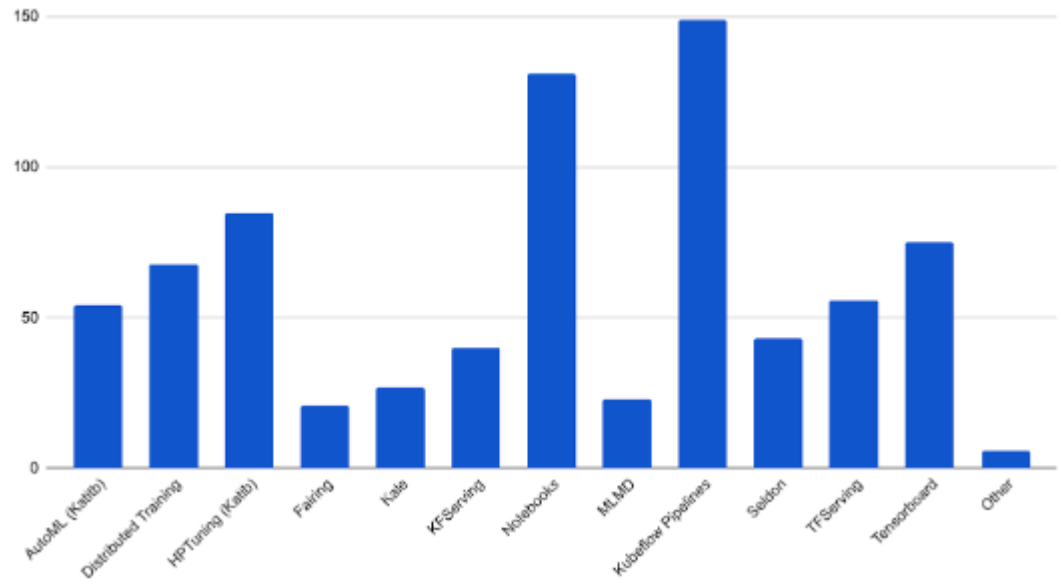



**Figura 19. Uso de las herramientas de KubeFlow**

Existen dos métodos para definir una canalización Kubeflow: mediante un notebook sencillo o mediante contenedores Docker. El proyecto Kale busca simplificar la experiencia de los científicos de datos al implementar flujos de trabajo de canalizaciones Kubeflow, ya que desarrollar y mantener estos flujos puede ser difícil para los científicos de datos que no son expertos en plataformas de orquestación. Kale resuelve esta brecha proporcionando una interfaz de usuario sencilla que permite etiquetar celdas de un *notebook* de Jupyter para definir flujos de trabajo de canalizaciones Kubeflow sin la necesidad de cambiar el código como podemos ver en la *Imagen 3*. A pesar de ser una forma fácil de definir una canalización Kubeflow, Kale no ofrece la misma flexibilidad y reutilización que se obtienen al utilizar contenedores Docker.

En cambio, al utilizar contenedores Docker, se puede lograr una canalización Kubeflow más modular y flexible. Cada tarea de aprendizaje automático se convierte en un contenedor Docker, lo que permite la portabilidad, reproducibilidad, encapsulamiento y reutilización de componentes en diferentes canalizaciones.

Además de las ventajas mencionadas, una canalización Kubeflow también permite la Integración Continua, Entrega Continua y Entrenamiento Continuo si se utiliza en conjunto con herramientas de CI / CD.

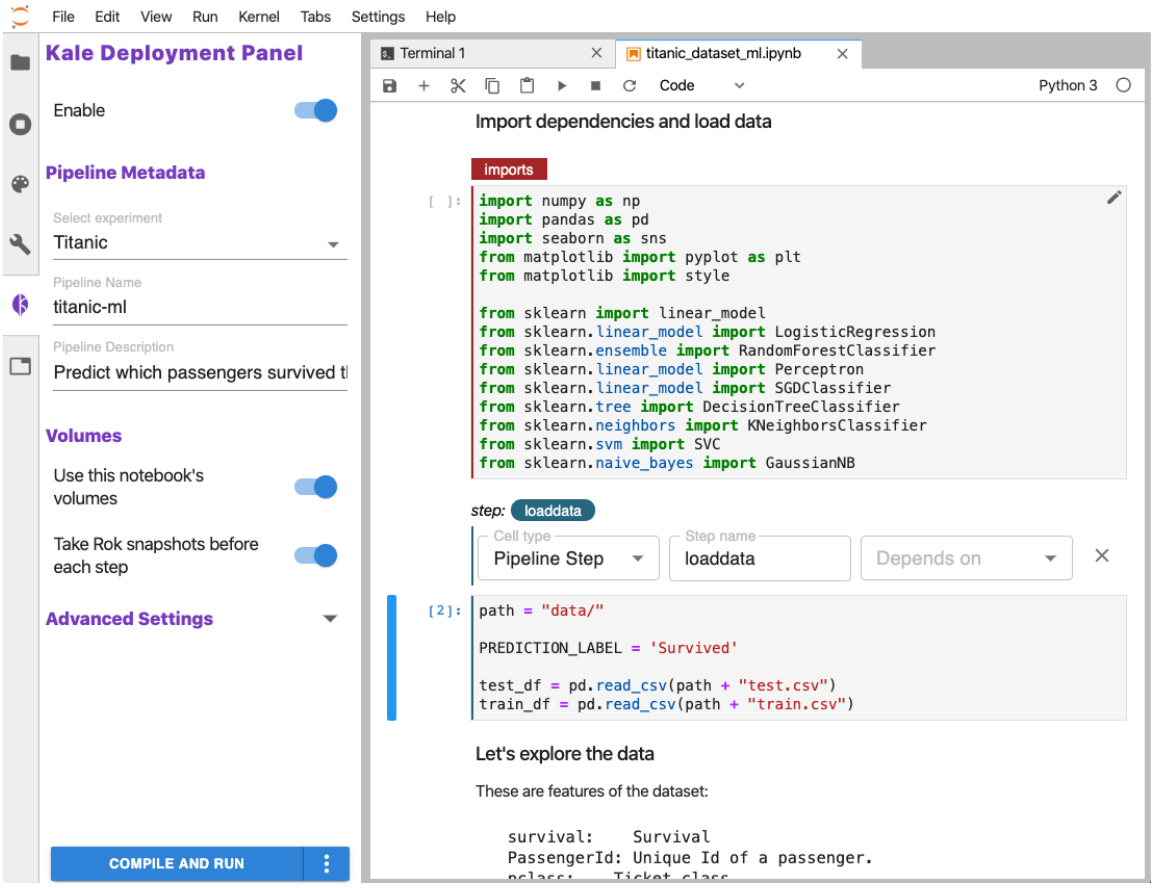



**Imagen 3. Herramienta Kale**

**Airflow**, una herramienta desarrollada por Airbnb, es una plataforma de propósito general que permite describir, ejecutar y monitorear flujos de trabajo. Los flujos de trabajo se definen en lenguaje Python para hacerlos más fáciles de mantener, versionar, probar y colaborar. No obstante, una limitación de Airflow es que las tareas no transfieren datos de una a otra, solo comparten metadatos. Los flujos de trabajo se representan como *DAGs* (Grafos Acíclicos Dirigidos) y pueden o no tener una planificación de ejecución definida. Airflow se puede integrar con numerosas plataformas, incluyendo AWS, Google Cloud Platform, Hadoop o Kubernetes entre otras. Otra de las principales ventajas que ofrece Airflow es el soporte de una gran comunidad, lo cual da la seguridad de que va a estar mantenida y el caso de tener cualquier problema lo podamos consultar.

Airflow cuenta con un amplio catálogo de operadores que nos permiten realizar multitud de tareas de forma sencilla, en el caso de que no exita un operador que se amolde a nuestras necesidades siempre existe la posibilidad de desarrollar uno a medida a partir de los predefinidos.

### 3.2.4. CI/CD

**Jenkins** se ha utilizado como herramienta de CI/CD durante mucho tiempo antes de la aparición de Kubernetes y los sistemas distribuidos en plataformas nativas de la nube. Sin embargo, trabajar con Jenkins puede resultar difícil. En los entornos en la nube, especialmente con la propagación de Kubernetes, ha surgido herramientas alternativas como Jenkins X siendo esta una opción para mejorar, automatizar, acelerar y simplificar las canalizaciones de integración y entrega continuas (CI/CD) en entornos en la nube como podemos ver en la *Figura 20*. Esto permite que los desarrolladores se centren en la creación de software. Jenkins se puede integrar con otros softwares de código abierto como Grafana (para registros centralizados y observabilidad), Tekton (para orquestación de canalizaciones nativas de la nube) y otras herramientas. Tanto Jenkins como Jenkins X se pueden utilizar para habilitar la CI/CD y la automatización. En MLOps, las dos herramientas se pueden adoptar para construir y ejecutar automáticamente las canalizaciones de aprendizaje automático.

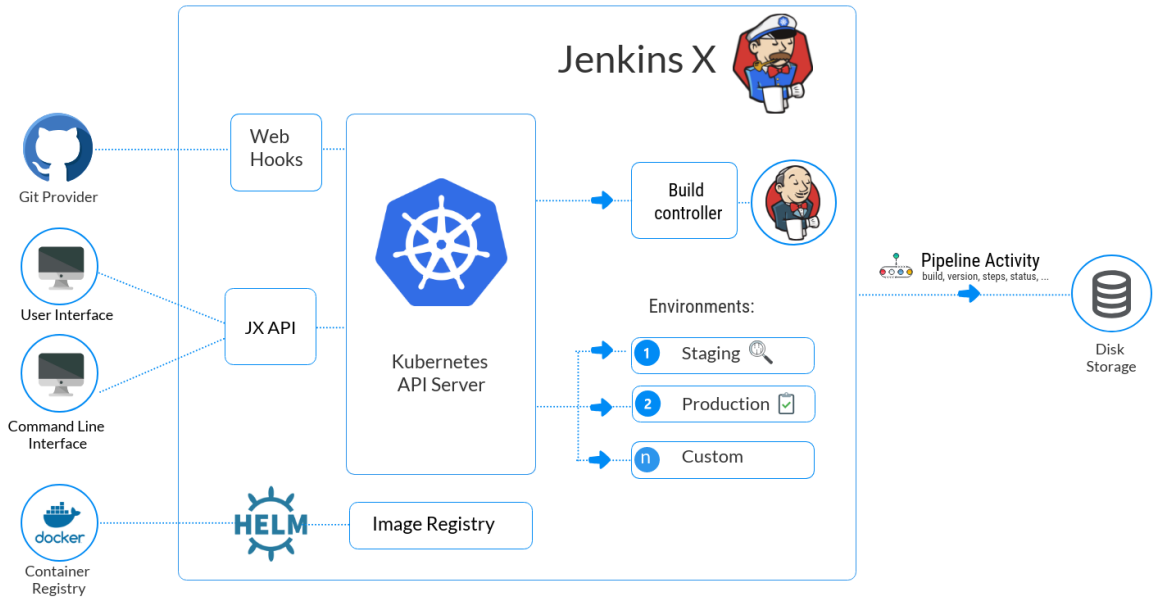



**Figura 20. Arquitectura y servicios de Jenkins X**

**Otras alternativas** muy populares a Jenkins son Github Actions y Gitlab CI/CD, ambas dos herramientas se encuentran integradas en los softwares de control de versiones de código más usados en la actualidad. Permiten integrar *plugins* que amplían su funcionalidad, pero no llegan a la potencia con que cuenta Jenkins.

### 3.2.5. Feature Store

Una variable, característica o "*feature*" se refiere a una propiedad que puede ser medida en los fenómenos observados y que se utiliza como parte de la entrada en un modelo de aprendizaje automático. Por otro lado, un "Feature Store" es una nueva capa de abstracción

que tiene como objetivo reducir el tiempo que los científicos de datos dedican a preparar los datos en un formato que puedan utilizar para entrenar modelos, permitiéndoles centrarse en la ciencia de datos. Esta herramienta actúa como un repositorio centralizado de diferentes variables, lo que permite su reutilización y compartición entre distintos equipos.

**Hopsworks** es una plataforma integral para el desarrollo *end-to-end* y la operación de aplicaciones de Machine Learning, creada por Logical Clocks. Su principal herramienta es su Feature Store, que es el software de código abierto más popular en este campo. La arquitectura de Hopsworks se muestra en la *Figura 21*. Su Feature Store simplifica el proceso de principio a fin proporcionando una API para que los ingenieros de datos creen variables y otra API para que los científicos de datos seleccionen fácilmente variables al diseñar nuevos modelos.

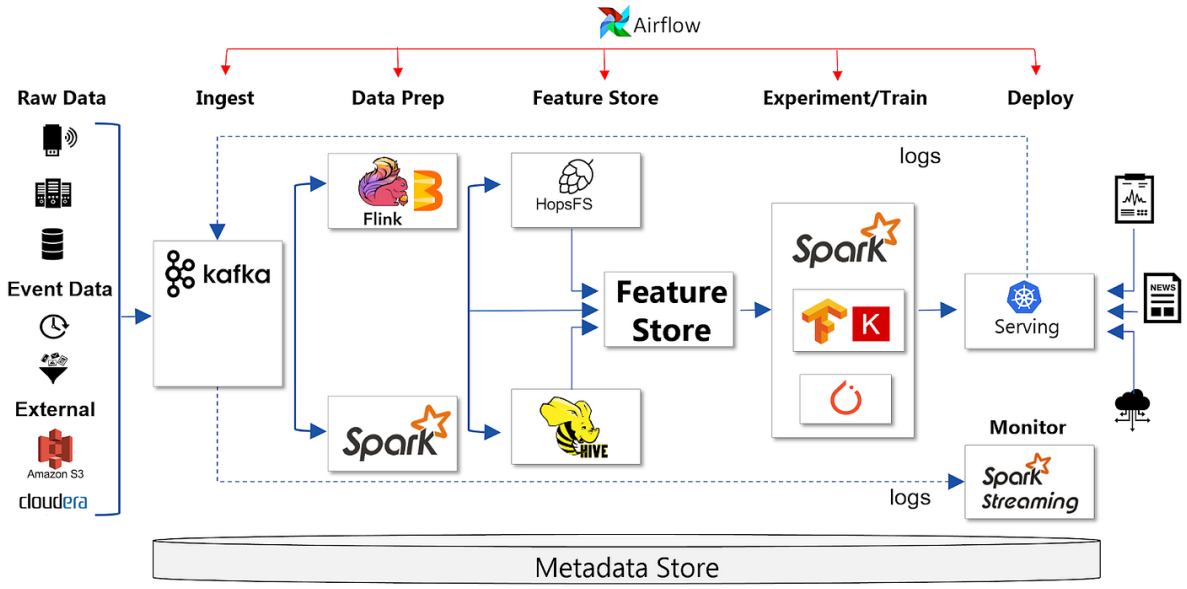



**Figura 21. Arquitectura Hopsworks**

### 3.2.6. Entrega/Despliegue (Serving)

**Seldon Core** es una plataforma de código abierto altamente efectiva para desplegar rápidamente modelos de aprendizaje automático en Kubernetes. La plataforma Seldon Core se encarga de escalar miles de modelos de aprendizaje automático a producción y proporciona diversas capacidades de ML listas para usar, como métricas avanzadas a través de Prometheus (una herramienta de monitoreo de código abierto), registro de solicitudes, explicadores (a través de su biblioteca de código abierto, Alibi), detección de valores anómalos, pruebas A/B, canarios, bandidos con múltiples brazos, entre otras como se puede ver en la *Figura 22* que muestra su arquitectura. Seldon Core puede integrarse con Kubeflow para administrar la implementación del sistema de aprendizaje automático desde

el orquestador de canalización, así como con Jenkins y JenkinsX para CI/CD entre otras herramientas.

**KServe** es un conjunto de herramientas de Kubeflow para implementar y prestar servicio a modelos, diseñado para superar los principales desafíos que se presentan al implementar modelos en producción. Ofrece inferencia sin servidor en Kubernetes y brinda interfaces de alto rendimiento y abstracción para los marcos de aprendizaje automático comunes como TensorFlow, XGBoost, scikit-learn, PyTorch y ONNX. KServe permite que los científicos de datos agreguen transformadores[9] y explicadores[10] al servidor central del modelo. Además, se puede incorporar un escalador automático al servicio de inferencia para observar el flujo de tráfico hacia la aplicación y ajustar el número de réplicas en Kubernetes en función de las métricas configuradas. Podemos ver la arquitectura de la herramienta en la *Figura 23*.

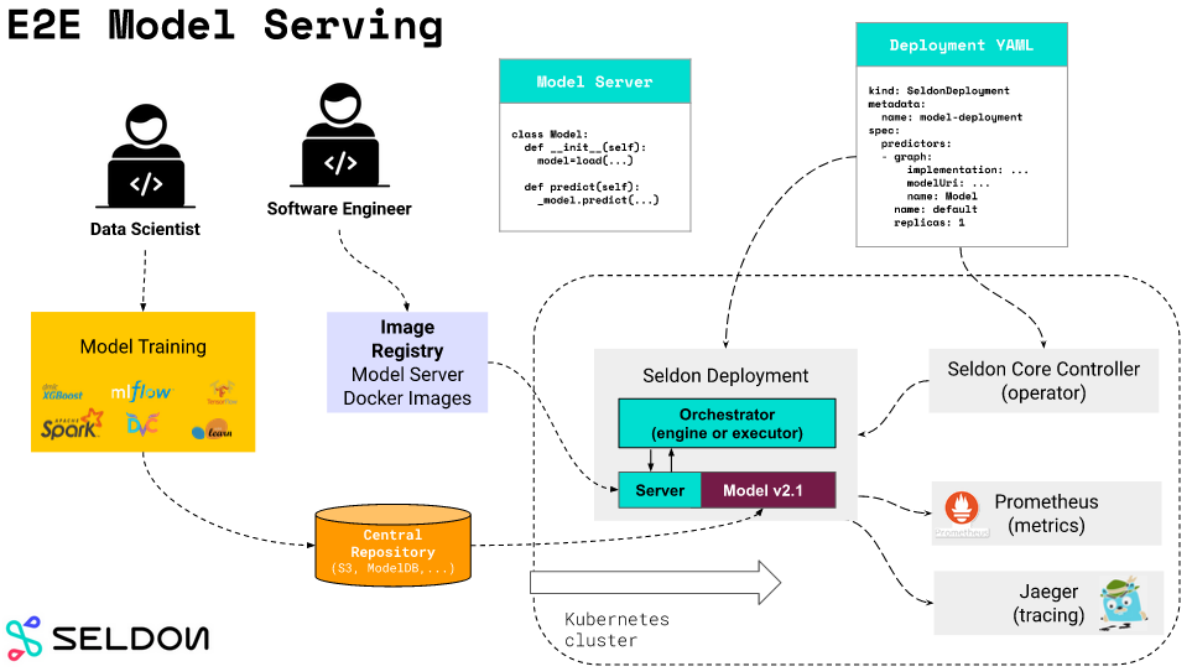



**Figura 22. Arquitectura Seldon Core**

**BentoML** es una herramienta flexible de alto rendimiento empleada para servir, administrar e implementar modelos de aprendizaje automático en producción. Se integra con varios marcos, como Tensorflow, PyTorch, Keras, XGBoost, entre otros, y se puede implementar de manera nativa en la nube utilizando herramientas como Docker, Kubernetes, AWS o

[9] Los transformadores permiten la transformación de datos de entrada y salida del modelo; por ejemplo, en un modelo de texto, es posible que se necesite transformar las palabras de entrada en vectores de embedding de palabras, los cuales constituyen la entrada cruda del modelo.

[10] Los explicadores son herramientas que permiten a los científicos de datos y auditores comprender el porqué de los resultados devueltos por el modelo permitiendo así comprenderlo.

Azure. BentoML ofrece un servicio de API en línea de alto rendimiento y también es compatible con un servicio de predicción por lotes *offline*.

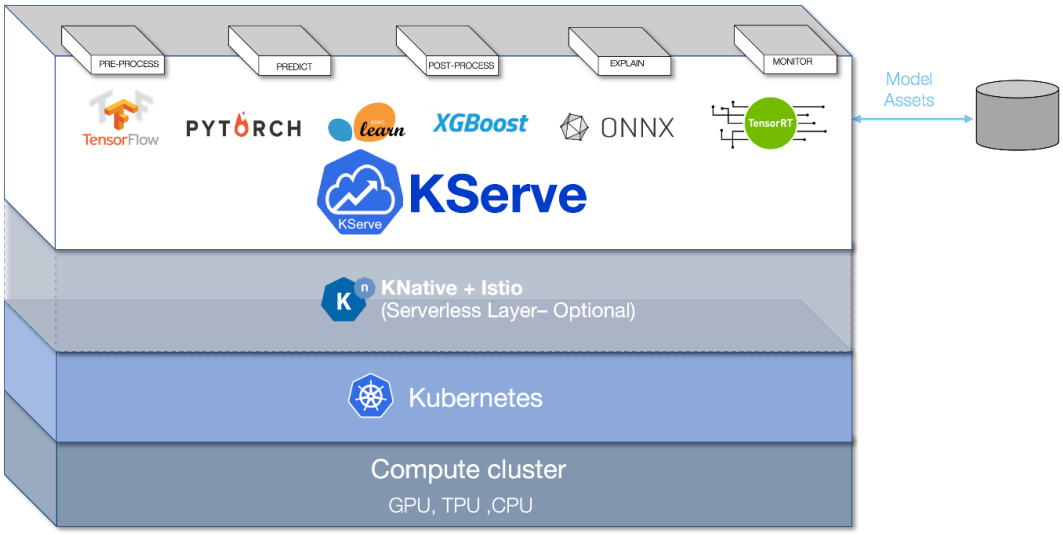



**Figura 23. Arquitectura KServe**

# 4. Implantar MLOps, Propuesta metodológica en el contexto actual

Es esencial construir una arquitectura efectiva que se adapte a las necesidades específicas de una empresa para aprovechar al máximo MLOps. Con la gran cantidad de herramientas disponibles, puede ser difícil determinar cuáles son las mejores tecnologías e infraestructuras a utilizar. Por lo tanto, es fundamental comprender el proceso empresarial, desde cómo se desarrollan los modelos hasta cómo se implementan en producción, para poder identificar qué herramientas y tecnologías se ajustan mejor a las necesidades empresariales. Como consecuencia de la constante evolución del espacio de MLOps, es crucial contar con una arquitectura técnica altamente modular que permita la flexibilidad para cambiar los bloques de construcción, herramientas y tecnologías en cualquier momento.

En el capítulo 3 se mencionan las principales herramientas de código abierto para MLOps haciendo hincapié en que los principales proveedores de la nube también ofrecen diferentes soluciones. No es necesario elegir exclusivamente entre herramientas de código abierto o soluciones en la nube, sino que se puede adoptar un enfoque híbrido. Las herramientas de código abierto brindan transparencia y flexibilidad en cuanto al proveedor de la nube, ya que suelen ser agnósticas en ese sentido. Además, algunas herramientas son consideradas "estándares de facto" en la actualidad, como Docker y Kubernetes; estas herramientas también permiten un enfoque modular. Por otro lado, las soluciones en la nube ofrecen un entorno completo que garantiza un soporte empresarial y, en general, ofrecen mejores herramientas que las específicas de código abierto. No obstante, las soluciones privativas de las nubes tienen desventajas, ya que reducen la extensibilidad y transparencia en una canalización y, al mismo tiempo, imponen un fuerte bloqueo del proveedor. Además, los servicios de los proveedores de la nube no son una solución viable para empresas que trabajan con dispositivos de software regulados o software con preocupaciones de privacidad de los usuarios, a no ser que lleguen a costosos acuerdos. En estos casos, se requiere una solución de MLOps que se pueda ejecutar en máquinas que sean agnósticas en cuanto a la nube y las instalaciones en red interna.

Se pueden encontrar plataformas de aprendizaje automático ofrecidas por proveedores de la nube como Google Cloud's Platform Vertex, Microsoft's AzureML y AWS's SageMaker para construir arquitecturas de MLOps. La elección de adoptar una plataforma de aprendizaje automático depende de la estrategia de la nube que tenga la organización y sus necesidades.

## 4.1. MLOps en el contexto empresarial

Implantar MLOps y crear un sistema de Machine Learning efectivo puede ser un gran reto para una empresa. Esto requiere cambiar la forma en que se desarrolla e implementa en el sistema, involucrar a diferentes equipos y presentar nuevas herramientas que se integren con los sistemas empresariales existentes, las opciones de la plataforma, la estrategia de canalización y las aplicaciones de monitoreo. Para incorporar un enfoque MLOps en el proceso empresarial, es fundamental diseñar una buena estrategia y enfocarse en la gestión del cambio requerido.

Existen dos enfoques diferentes a adoptar, los cuales pueden combinarse, según las habilidades y experiencia de las personas involucradas, los requisitos del cliente y la infraestructura en uso. El primer enfoque se centra en la complejidad de las herramientas y el segundo en el nivel de automatización. Sin importar el enfoque, es esencial hacer un diagnóstico de las prácticas y procesos actuales mediante la organización de múltiples entrevistas con las partes clave interesadas.

El método más simple implica introducir tecnologías MLOps gradualmente, comenzando con la herramienta más sencilla, como MLFlow. Esto permite que los equipos experimenten los beneficios de las mejores prácticas de MLOps y se familiaricen con la nueva metodología. Luego, se pueden priorizar herramientas que brinden un mayor valor, como Kubeflow, para garantizar la reproducibilidad y la automatización.

Un enfoque más avanzado implica mejorar el nivel de automatización en el proceso de construcción del sistema de Machine Learning. Como se ha comentado en apartados anteriores, se identifican tres niveles de automatización, y el último nivel implica la introducción de un sistema completo de CI/CD.

## 4.2. Comparativa y análisis de las principales herramientas MLOps

Este apartado busca analizar que herramienta elegir para satisfacer los requisitos de una arquitectura MLOps, el entorno y la contenerización, el registro de experimentos, la orquestación de pipelines, el CI/CD, un *Feature store,* así como el despliegue de modelos de aprendizaje automático.

Para realizar dicho análisis se seleccionarán las herramientas más populares de cada uno de los ámbitos identificados. Se analizarán tanto herramientas gratuitas como las que proponen los principales proveedores en la nube como Google Cloud's Platform Vertex, Microsoft's AzureML y AWS's SageMaker. Se condensarán las principales características de las herramientas en tablas que permitan de forma ágil y sencilla conocer las bondades de cada una de ellas. Con esta información se podrá seleccionar la herramienta que se adapte mejor a las necesidades específicas de la empresa.

Estas tablas contarán con una serie de campos que se compartirán entre todas ellas, así como alguno específico a cada uno de los ámbitos a cubrir. Los campos comunes son:

- **Enlace** a la herramienta.
- **Año** de publicación de la herramienta. Este dato nos permitirá hacernos una idea de la madurez de la herramienta además nos ofrecerá una visión sobre su asentamiento en el contexto empresarial actual.
- **Open-source**, si son herramientas de código abierto.
- **Alojado**, nos indica si las herramientas se encuentran en la nube o por el contrario son de uso local.
- **Gestionado**, si debemos de encargarnos de mantener la aplicación desplegada o por el contrario nos es transparente.
- **Interfaz de uso**, nos indica como el usuario pude interactuar con la herramienta. Hay dos posibilidades: CLI (por consola de comandos) y GUI (interfaz gráfica).
- **Comunidad y soporte**, nos indica si la herramienta cuenta con una comunidad activa en la que encontrar o preguntar la solución a los problemas que puedan surgir.
- **Ventajas frente a los competidores**, detalla los puntos clave por los que elegir una herramienta frente a otra.
- **Madurez**, nos indica el grado de madurez de una herramienta. Este es un dato clave a tener en cuenta a la hora de elegir una herramienta, ya que una con un bajo grado de madurez es probable que tenga mayor número de fallos que las que ya están asentadas.
- **Curva de aprendizaje**, este es un dato clave a la hora de implantar una herramienta. Nos indica cuanto tiempo requiere la herramienta para ser dominada por los usuarios. En muchos casos este factor ha sido clave para el fracaso o éxito de la implantación de nuevas herramientas.

### 4.2.1. Entorno y Contenerización

En este apartado se analizarán las herramientas: Docker, Podman, VirtualBox, Kubernetes, Docker Swarm, Nomad. Algunas de estas herramientas permiten empaquetar el software en contenedores estándar que contienen todo lo necesario para su ejecución, incluyendo bibliotecas, herramientas de sistema y código en tiempo de ejecución. Las otras son herramientas para la orquestación de los contenedores que permiten gestionarlos. La *Tabla 1* contiene un resumen de la tabla de resultados del análisis que podemos encontrar en el *Anexo Entorno y contenerización*.

Todas las herramientas mencionadas son gratuitas, aunque algunas ofrecen opciones de pago para obtener características adicionales y soporte empresarial. Docker y Nomad

ofrecen versiones empresariales (Docker Enterprise y Nomad Enterprise, respectivamente) que incluyen características adicionales, soporte y opciones de gestión avanzadas para empresas. Kubernetes, por otro lado, es gratuito, pero muchos proveedores de la nube ofrecen servicios gestionados basados en Kubernetes que tienen costos asociados. En general, estas herramientas son accesibles y económicas para la mayoría de los usuarios y organizaciones.

En la actualidad, la combinación de Docker y Kubernetes es la opción más popular para entornos y contenerización. Docker es una herramienta relativamente sencilla que permite generar entornos desde la fase de desarrollo hasta la fase de despliegue, asegurando que se ejecuten en el mismo entorno en todos los casos. Esto cumple con el principio de reproducibilidad y evita los problemas de "En mi local funciona", frase muy común entre los desarrolladores. Docker cuenta con un alto grado de madurez y una amplia comunidad que puede brindar soporte en caso de problemas.

Kubernetes, el orquestador de contenedores más popular para Docker, también cuenta con un alto grado de madurez y una comunidad de desarrolladores muy activa. Ambas herramientas son estándares en las principales nubes del mercado, lo que las convierte en la mejor opción frente a sus principales competidores para entornos y contenerización.

| Herramienta | Año | Open-Source | Alojado | Gestionado | Tipo de virtualización | Madurez | Precio |
|---|---|---|---|---|---|---|---|
| **Docker** | 2013 | Sí | Local o en la nube | Sí | Contenedores | Madura | Gratis |
| **Podman** | 2018 | Sí | Local o en la nube | No | Contenedores | Emergente | Gratis |
| **VirtualBox** | 2007 | Sí | Local | No | Máquinas virtuales | Madura | Gratis |
| **Kubernetes** | 2014 | Sí | En la nube o en local | Sí | Contenedores | Madura | Gratis |
| **Docker Swarm** | 2015 | Sí | En la nube o en la máquina huésped | Sí | Contenedores | Madura | Gratis |
| **Nomad** | 2015 | Sí | Local o en la nube | Sí | Contenedores | Emergente | Gratis |

**Tabla 1. Análisis de herramientas de entorno y Contenerización**

### 4.2.2. Registro de experimentos

En este apartado se anlizan las herramientas: Mlflow, Neptune, Azure Machiner Learning Model Registry, Vertex Model Registry, Sagemaker Model Registry. Estas herramientas permiten registrar diferentes metadatos de cada uno de los entrenamientos de modelos ML. Entre estos datos se registran métricas, hiperparámetros, otros datos, así como artefactos y modelos. Con toda esta información se puede trazar la evolución de los modelos en cuanto a métricas e hiperparámetros, lo que permite reproducir el entrenamiento de los modelos además de recuperar los modelos asociados a cada experimento, así podemos tenerlos almacenados de forma centralizada. En la *Tabla 2* encontramos un resumen de la tabla de resultados del análisis que podemos encontrar en el *Anexo Registro de experimentos*.

Tras analizar los datos recopilados en la tabla, elegiremos MLFlow como herramienta de registro de modelos por varias razones. En primer lugar, MLFlow es una herramienta de código abierto que permite registrar el ciclo de vida desde el entrenamiento hasta la implementación de los modelos. En segundo lugar, es compatible con numerosas bibliotecas y *frameworks* de machine learning, lo que permite una fácil integración con otras herramientas del ecosistema de ML. En tercer lugar, cuenta con diversas funcionalidades, como la trazabilidad de los experimentos de entrenamiento y la gestión de versiones de los modelos entre otras. En cuanto a la seguridad, MLflow ofrece una opción de autenticación y autorización basada en tokens para controlar el acceso a los modelos y sus metadatos. Al ser una herramienta de código abierto, existe una comunidad activa de desarrolladores de MLflow que pueden colaborar en la solución de cualquier problema de que surja en la herramienta.

Pese a que todas las herramientas tienen un funcionamiento similar, tienen ligeras diferencias que podrían llevarnos a preferir una sobre las otras. Si usamos una plataforma en la nube como Microsoft, Amazon o Google, optaríamos por la herramienta nativa propia de la nube, ya que no existen grandes diferencias entre ellas.

| Herramienta | Año | Open-Source | Alojado | Lenguajes compatibles | Madurez | Integración con herramientas | Precio |
|---|---|---|---|---|---|---|---|
| **Mlflow** | 2018 | Sí | On-premises o en la nube | Python, R, Java | Estable | Fácilmente | Gratis |
| **Neptune** | 2018 | No | En la nube | Python, R, Java, Scala | Estable | Fácilmente | Pago |
| **Azure Machine Learning Model Registry** | 2019 | No | En la nube | Python, R, Java | Estable | Nativa | Pago |
| **Vertex Model Registry** | 2019 | No | En la nube | TensorFlow, PyTorch, scikit-learn, XGBoost | Estable | Nativa | Pago |
| **Sagemaker Model Registry** | 2017 | No | En la nube | TensorFlow, PyTorch, scikit-learn, XGBoost | Estable | Nativa | Pago |

**Tabla 2. Análisis de herramientas de registro de experimentos**

### 4.2.3. Orquestación de pipelines

En este apartado analizaremos las herramientas: Kubeflow, Airflow, Azure Machine Learning Pipelines, Vertex Pipelines y Amazon SageMaker Pipelines. Estas herramientas permiten la definición de flujos de trabajo en forma de grafo donde cada nodo se corresponde con un paso que se ejecuta de forma independiente y aislada con dependencias de los pasos anteriores. Con esta estructura lo que conseguimos es modularidad e independencia de entornos entre pasos lo que nos aporta ciertas ventajas. Entre estas ventajas encontramos la facilidad de mantenimiento ya que podemos sustituir, añadir o borrar pasos de forma sencilla. Otra ventaja es que cada paso pude tener requerimientos tanto de software como de recursos diferentes, lo cual podemos gestionar ejecutándolos en contenedores diferentes. En la *Tabla 3* encontramos un resumen de la tabla de resultados del análisis que podemos encontrar en el Anexo *Orquestación de Pipelines*.

Tras analizar los datos recogidos escogeríamos la herramienta prestando especial atención al nivel técnico del equipo que fuera a emplearla. Si estuviéramos frente a un equipo con

un conocimiento avanzado de programación y contase con algo de experiencia previa con el desarrollo de pipelines elegiríamos Kubeflow. En el caso de que el equipo no tuviera un conocimiento en programación tan avanzado escogería la opción de Airflow, ya que cuenta con características similares y es más simple para desarrollar los pipelines. Además, Airflow está disponible en las tres nubes.

Kubeflow es una plataforma de código abierto alojada en la nube, diseñada específicamente para automatizar el despliegue, la gestión y el seguimiento de flujos de trabajo de aprendizaje automático en Kubernetes. Una de las principales ventajas de Kubeflow es que proporciona una gran cantidad de integraciones con diferentes herramientas y frameworks de aprendizaje automático, lo que la hace muy versátil y flexible. Otra ventaja de Kubeflow es su escalabilidad. Al estar alojada en Kubernetes, puede escalar automáticamente el hardware y la capacidad de procesamiento según sea necesario. Además, es una herramienta madura que cuenta con una gran comunidad de usuarios y desarrolladores los cuales proporcionan soporte y actualizaciones regulares.

| Herramienta | Año | Open-Source | Alojado | Gestionado | Características principales | Curva de aprendizaje | Precio |
|---|---|---|---|---|---|---|---|
| **Kubeflow** | 2017 | Sí | On-premises o en la nube | No | Integración con TensorFlow, PyTorch, XGBoost, entre otros. | Media | Gratis |
| **Airflow** | 2015 | Sí | On-premises o en la nube | No | Integración con Hadoop, Spark, Hive, entre otros. | Baja | Gratis |
| **Azure Machine Learning Pipelines** | 2019 | No | En la nube (Azure) | Sí | Integración con herramientas de Azure, como Azure Machine Learning Studio. | Media | Pago |
| **Vertex Pipelines** | 2020 | No | En la nube (Google Cloud) | Sí | Integración con herramientas de Google Cloud, como Vertex AI. | Baja | Pago |
| **Amazon SageMaker Pipelines** | 2020 | No | En la nube (AWS) | Sí | Integración con herramientas de AWS, como SageMaker. control de versiones. Gran seguridad y privacidad | Baja | Pago |

**Tabla 3. Análisis de herramientas de orquestación pipelines**

### 4.2.4. CI/CD

En este apartado se analizarán las herramientas: Jenkins, Azure Pipelines, Cloud Build y AWS CodePipeline. Estas herramientas permiten realizar toda la integración y entrega continuas. Como se ha comentado anteriormente esta es una parte crucial para poder implementar de forma satisfactoria la metodología MLOps. Todas ellas permiten integración con repositorios, servicios en la nube, herramientas de terceros y soporte para múltiples plataformas. En la *Tabla 4* encontramos un resumen de la tabla de resultados del análisis que podemos encontrar en el *Anexo CI/CD*.

La herramienta de CI/CD más popular y extendida desde hace años, así como la más madura es Jenkins. Es la única de código abierto de las analizadas. Si vamos a trabajar en una de las nubes, elegiría la nativa de esta nube. En el caso de que existiera alguna necesidad específica que no estuviera cubierta sí que plantearíamos el uso de Jenkins.

| Herramienta | Año | Open-Source | Alojado | Configuración y personalización | Madurez | Curva de aprendizaje | Precio |
|---|---|---|---|---|---|---|---|
| **Jenkins** | 2004 | Sí | On-premises o en la nube | Alta | Muy maduro | Alta | Gratis |
| **Azure Pipelines** | 2018 | No | En la nube (Azure) | Alta | Maduro | Media | Pago |
| **Cloud Build** | 2018 | No | En la nube (Google Cloud) | Media | En desarrollo | Baja | Pago |
| **AWS CodePipeline** | 2015 | No | En la nube (AWS) | Media | Maduro | Media | Pago |

**Tabla 4. Análisis de herramientas de CI/CD**

### 4.2.5. Feature Store

En este apartado se analizarán las herramientas: Hopsworks, Azure Databricks, Vertex Feature Store y Amazon SageMaker Feature Store. Estas herramientas como se ha comentado en apartados anteriores, que permiten almacenar y registrar a modo de catálogo todas las variables que usemos en el entrenamiento de los modelos. Nos permite reutilizar variables y agilizar el desarrollo de nuevos modelos ya que muchas de las variables a emplear pueden estar ya creadas. En la *Tabla 5* encontramos un resumen de la tabla de resultados del análisis que podemos encontrar en el *Anexo Feature Store*.

La elección de una herramienta como *Feature Store* dependerá de en qué contexto nos encontremos, es decir, si no estamos en un entorno en la nube usaremos la herramienta *open-source* (Hopsworks), en el caso de estar en una nube usaremos la herramienta nativa de *Feature Store*.

| Herramienta | Año | Open-Source | Alojado | Lenguajes compatibles | Base de datos | Madurez | Curva de aprendizaje | Precio |
|---|---|---|---|---|---|---|---|---|
| **Hopsworks** | 2018 | Sí | On-premises o en la nube | Python, Java, Scala | HopsFS | Madura | Moderada | Pago |
| **Azure Databricks** | 2018 | No | En la nube | Python, R, Scala, SQL | Azure Data Lake Storage, Azure Blob Storage | Madura | Moderada | Pago |
| **Vertex Feature Store** | 2021 | No | En la nube | Python, Java, Go, C++, Rust, TensorFlow | Google Cloud Storage, BigQuery | Madura | Moderada | Pago |
| **Amazon SageMaker Feature Store** | 2020 | No | En la nube | Python, R, Java, Scala, TensorFlow, PyTorch, MXNet | Amazon S3, Amazon Redshift, Amazon DynamoDB | Madura | Moderada | Pago |

**Tabla 5. Análisis de herramientas de Feature Store**

### 4.2.6. Entrega/Despliegue (Serving)

En este apartado se analizarán las herramientas: Seldon Core, KServe, BentoML, Azure Machine Learning Endpoints, Vertex Endpoint y SageMaker Endpoint. Estas herramientas permiten agilizar la implantación de los modelos ML en producción mediante una API la cual podemos usar para obtener predicciones realizadas por ese modelo. En la *Tabla 6* encontramos un resumen de la tabla de resultados del análisis que podemos encontrar en el *Anexo Entrega/Despliegue (Serving)*.

Analizando las características principales de las herramientas para el despliegue de modelos ML, se recomienda KServe como herramienta para el despliegue de modelos. Aunque todas las herramientas tienen un funcionamiento similar, con ligeras diferencias que podrían llevarnos a preferir una sobre otra. Si usamos una plataforma en la nube, optaríamos por la herramienta nativa de la nube, ya que no existen grandes diferencias entre ellas.

| Herramienta | Año | Open-Source | Alojado | Lenguajes compatibles | Integración con plataformas de nube | Monitoreo | Madurez | Curva de aprendizaje | Precio |
|---|---|---|---|---|---|---|---|---|---|
| **Seldon Core** | 2018 | Sí | On-premise, serverless | Python, Java, R | Kubernetes, Amazon EKS, Google Kubernetes Engine | Prometheus, Grafana | Estable | Moderada | Gratis |
| **KServe** | 2019 | Sí | On-premise, serverless | Python, Java, Node.js | Kubernetes | Prometheus, Grafana | Estable | Moderada | Gratis |
| **BentoML** | 2020 | Sí | On-premise, serverless | Python | Kubernetes, Docker, AWS Lambda, Google Cloud Functions, Azure Functions | Prometheus, Grafana | Estable | Moderada | Gratis |
| **Azure Machine Learning Endpoints** | 2018 | No | En la nube (Azure) | Python, R | Azure | Azure Monitor, Application Insights | Estable | Fácil | Pago |
| **Vertex Endpoint** | 2020 | No | En la nube (Google Cloud) | Python | Google Kubernetes Engine | Cloud Monitoring, Cloud Trace | Estable | Fácil | Pago |
| **SageMaker Endpoint** | 2017 | No | En la nube (AWS) | Python | Amazon SageMaker | Amazon CloudWatch | Estable | Fácil | Pago por uso |

**Tabla 6. Análisis de herramientas de Entrega/Despliegue**

## 4.3. Elección de entorno

A la hora de montar el entorno MLOps debemos elegir que infraestructura emplear según las necesidad y requisitos de negocio. Existen múltiples factores que pueden decantarnos por una opción u otra, entre las que se encuentran el factor económico, el grado de madurez de la plataforma o casuísticas particulares.

Lo primero que deberíamos plantear es si queremos un entorno *On-premisse* o en la nube. Para ello se deben evaluar factores cruciales a la hora de elegir una nube, como son la tipología de datos a tratar, si existen datos que por los reglamentos de datos como la RGPD haya que cumplir con una serie de requisitos más restrictivos. Si no disponemos del capital suficiente para convertirnos en *partners* de alguna nube que permitan tener los servidores de ejecución dentro de la red interna de la empresa, la mejor opción es tener un entorno *On-premise*. Este es el caso por ejemplo de los bancos, los cuales deben de asociarse con nubes para que los datos no salgan de su red interna.

Si nos decantamos por un entorno en la nube deberemos de elegir la nube que más se adapte a nuestras necesidades. Según un informe de la consultora Gartner del año 2022 podemos ver el posicionamiento de las diferentes nubes (Gartner, 2022). En la Figura *24* podemos ver el cuadrante mágico, como ellos lo llaman, donde cuanto más se aproximen a la esquina superior derecha mejor. Las dimensiones que se analizan en el Magic Quadrant son las siguientes, donde vemos que las mejor posicionadas son la nube de Amazon (AWS), seguida por la de Microsoft (Azure) y la de Google (GCP):

- **Líderes**. Los líderes se destacan al proporcionar un servicio adecuado para la implementación estratégica y tener una visión ambiciosa a seguir. Son capaces de atender una amplia variedad de situaciones de uso, aunque no sobresalen en todos los aspectos. Es importante destacar que no necesariamente son los mejores proveedores para una necesidad particular y es posible que no ofrezcan servicios para algunos casos de uso en particular. Los líderes en este mercado cuentan con una participación significativa en el mercado y tienen numerosos clientes de referencia.
- **Contendientes**. Los contendientes gozan de una buena posición para satisfacer ciertas necesidades actuales del mercado. Su servicio es sólido y se enfoca en un conjunto específico de casos de uso, además de contar con un historial exitoso de entregas. Sin embargo, no están respondiendo lo suficientemente rápido a los desafíos del mercado o carecen de una visión amplia y ambiciosa.
- **Visionarios**. Los visionarios tienen una perspectiva ambiciosa del futuro y realizan inversiones significativas en el desarrollo de tecnologías innovadoras. Aunque sus servicios se encuentran en fases emergentes, están trabajando en numerosas capacidades en desarrollo que aún no están disponibles para todos. Pese a que pueden contar con numerosos clientes, es posible que aún no brinden los servicios

adecuados para una amplia variedad de casos de uso o que su alcance geográfico sea limitado.

- **Especialistas**. Los especialistas en el mercado de CIPS (Cloud Infrastructure and Platform Services) pueden ser proveedores destacados para situaciones de uso específicas o en las regiones en las que operan, sin embargo, deben ser considerados principalmente como proveedores especializados. En general, no ofrecen servicios adecuados para una amplia variedad de casos de uso ni tienen una hoja de ruta altamente ambiciosa. Algunos pueden tener posiciones de liderazgo sólidas en mercados relacionados, pero su enfoque en capacidades de CIPS es limitado.

La recomendación de qué nube a elegir puede verse afectada por características subjetivas, por ejemplo, si somos una empresa de seguros quizá AWS no queramos que sea nuestro proveedor puesto que también venden seguros. Siendo puramente objetivos la elección de nube más recomendable actualmente es AWS seguida de GCP por las características detalladas en los siguientes subapartados. Azure es una alternativa que, si bien actualmente es la segunda mejor posicionada, sus perspectivas a futuro no son muy halagüeñas.

### 4.3.1. Amazon Web Services (AWS)

Según el informe de la consultora Gartner, es el proveedor en la nube de referencia actualmente. Es un proveedor global cuyos clientes abarcan diferentes perfiles, tamaños, sectores y localizaciones. Entre sus puntos fuertes destacan:

- El gran catálogo de funcionalidades que ofrece, siendo el proveedor con mayor oferta y profundidad de capacidades. Es quien guía al resto de proveedores, marcando el camino a seguir, estableciendo estándares y desarrollando tecnologías y metodologías que a menudo copian sus competidores.
- Líder del mercado, siendo el proveedor que mayor cuota de mercado cubre.

### 4.3.2. Google Cloud Platform (GCP)

Según el Magic Quadrant Google es el tercer proveedor en el ranking. Ha conseguido ser robusto en casi todos los casos de uso posibles con una gran evolución y progresión que ha mejorado sus capacidades. Entre sus puntos fuertes destacan:

- Aumento de capacidades e ingresos. GCP logró el porcentaje más alto de ganancias y mejoras en los ingresos.
- Cambio en el enfoque de ventas. El cambio hacia la venta a ejecutivos de negocios está teniendo resultados notables, tanto en términos de adopción como de presencia comercial.
- Nube soberana. Fuera de sus regiones centrales, el GCP tiene una postura de asociación más relajada. Está dispuesto a trabajar con operadores nacionales para crear alternativas de nube en regiones clave.

### 4.3.3. Microsoft Azure

Según el informe, Microsoft Azure es el segundo proveedor en el ranking. Al igual que GCP es robusto en todos los casos de uso con un claro enfoque en nubes híbridas.

- Cuota de mercados. Azure está cerrando la brecha de mercado con AWS. En un futuro previsible, la brecha global entre los dos se reducirá significativamente, hecho que ya está ocurriendo en Europa.
- Orientada a soluciones. Microsoft y sus socios tienen una amplia gama de capacidades en la nube que ayudan a cumplir con los casos de uso de los clientes. Microsoft tiene una posición diferenciada en segmentos amplios como telecomunicaciones, salud, manufactura y venta minorista.
- Multinube híbrida. Azure apuesta por la creencia de que la mayoría de las empresas usarán nubes híbridas y multinube, las cuales necesitan operaciones y gobernanza simplificadas para operar entornos tan diversos de forma segura y eficaz.

Sin embargo, es interesante mencionar los problemas de seguridad que ha tenido Azure en el pasado además de la falta de innovación provocada por la necesidad de solventarlos.


**Figura 24. Magic Quadrant para servicios de infraestructura y plataforma en la nube**

# 5. Conclusiones y Vías de continuación

En los últimos años ha habido un auge de la inteligencia artificial tanto en la industria como en la sociedad, siendo el mayor ejemplo de ello el popular modelo Chat GPT. Cada vez se desarrollan más modelos los cuales por la velocidad con que cambia el contexto socioeconómico pueden perder valor rápidamente. Es por este motivo, que se necesita una metodología como MLOps que permita automatizar y agilizar la puesta en producción de nuevos modelos, permitiendo aumentar su valor generado. Los retrasos en su puesta en producción pueden suponer grandes pérdidas económicas, ya que todo el trabajo de desarrollo de los modelos puede ser en vano si no se ponen en producción con la suficiente agilidad.

El objetivo principal de este trabajo ha sido el análisis de la metodología MLOps en el contexto actual, así como su evolución en el futuro, observando las iniciativas y soluciones que van surgiendo tanto en la comunidad *Open-Source* como en los principales proveedores de la nube. Con este análisis se ha pretendido estudiar las bondades y limitaciones a tener en cuenta a la hora de aplicar esta metodología junto con un análisis de las soluciones actuales disponibles.

Como se ha expuesto durante el trabajo, MLOps surgió a partir de la metodología DevOps con la idea de poder simplificar en gran medida el flujo de trabajo de Aprendizaje Automático y acelerar todo el proceso de llevar los modelos a producción. Para lograr este objetivo, se pueden adoptar técnicas como CI/CD, la Capacitación y Monitoreo Continuos, la automatización, etc.; garantizando la reproducibilidad, diseñando el sistema de Aprendizaje Automático de forma modular y aplicando todas las demás características de MLOps.

Los principales beneficios de aplicar esta metodología son, como ya se ha comentado anteriormente, la agilidad en la puesta en producción de los modelos de aprendizaje automático, la reproducibilidad de los resultados y las ejecuciones, junto con una mejora en la escalabilidad. La mayoría de estas ventajas vienen propiciadas por la automatización de los pasos del flujo de desarrollo. Por el contrario, nos encontramos con ciertos inconvenientes como la elevada curva de aprendizaje para los desarrolladores que no han trabajado previamente en entornos similares, lo que puede suponer un rechazo inicial a esta metodología. En este sentido, no solo es importante valorar el nivel de conocimiento de los equipos de trabajo sino también determinar el grado de madurez de la compañía, lo cual nos ayudará a elegir el nivel de MLOps a implantar en un primer momento para poder evolucionar más adelante hasta conseguir tener un entorno MLOps al 100%. Por último, también hay que valorar el impacto económico inicial que supone implementar toda la infraestructura necesaria.

En la última parte del estudio hemos analizado las tendencias a futuro y como la plantean los principales proveedores en la nube como son Amazon, Microsoft o Google. Tras analizar

sus propuestas observamos que todos ellos ofrecen prácticamente los mismos servicios con sutiles diferencias. En el caso de Google, éste estrenó hace menos de un año su plataforma dedicada al MLOps, Vertex, lo que nos hace suponer que esta metodología va a seguir creciendo de forma acelerada. También analizamos las diferentes soluciones disponibles de código abierto que nos permiten desarrollar una solución *On-Premise*.

La conclusión que extraemos tras realizar este trabajo es que MLOps es una metodología en auge en el ámbito de la inteligencia artificial, la cual seguirá creciendo y evolucionando para satisfacer las diferentes necesidades que pudieran surgir. Así lo ven los grandes proveedores de entornos en la nube o grandes consultoras como Gartner. También podemos afirmar que no existe una solución óptima, dependerá de la situación en la que nos encontremos. Como comentado anteriormente, la elección del entorno se verá condicionada por múltiples factores como son la nube en la que trabaja la empresa o la naturaleza de los datos.

Como continuación de este trabajo sería interesante poder investigar y probar a fondo todas las herramientas analizadas en este trabajo, así como otras que surjan en el futuro. También habría que monitorizar los cambios y mejoras que pudieran ir implementando los distintos proveedores de servicios en la nube de manera que podamos comprobar si las expectativas actuales sobre ellas se cumplen, que nuevos enfoques MLOps adoptan y si dichos enfoques son capaces de mejorar la metodología actual.

# 6. Bibliografía

*Aws MLOps Framework*. (s.f.). Obtenido de https://aws.amazon.com/es/solutions/implementations/mlops-workload-orchestrator/; https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-projects-why.html

*Azure MLOps Framework*. (s.f.). Obtenido de https://learn.microsoft.com/en-us/azure/machine-learning/concept-model-management-and-deployment?view=azureml-api-2

Bala, R., Smith, D., Ji, K., Wright, D., & Borrega , M. Á. (19 de Octubre de 2022). *Gartner*. Obtenido de https://www.gartner.com/technology/media-products/reprints/AWS/1-2AOZQARI-ESL.html

Chilimbi, T., Suzue, Y., Apacible, J., & Kalyanaraman, K. (2014). Project adam: Building an efficient and scalable deep learning training system. *OSDI'14: Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation*, (págs. 571–582). Broomfield, CO, USA.

*Continuous Delivery*. (s.f.). Obtenido de agilealliance: https://www.agilealliance.org/agile101/agile-basics/continuous-delivery//

*DataOps Manifesto*. (s.f.). Obtenido de https://dataopsmanifesto.org/

*Google. MLOps: Continuous Delivery and automation pipelines in Machine*. (s.f.). Obtenido de https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

Li, M., Andersen, D., Park, J., Smola, A., Ahmed, A., Josifovski, V., . . . Su, B.-Y. (2014). Scaling Distributed Machine Learning with the Parameter Server. *OSDI'14: Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation*, (págs. 583–598). Broomfield, CO, USA.

Robinson, S., Lakshmanan, V., & Munn, M. (2020). *Machine Learning Design Patterns.* O'Reilly Media, Inc.

Ruiz Cristina, J. (16 de 11 de 2015). *El legendario origen del movimiento devops*. Obtenido de https://www.paradigmadigital.com/techbiz/el-legendario-origen-del-movimiento-devops/

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., . . . Dennison, D. (2014). Hidden Technical Debt in Machine Learning Systems. *SE4ML workshop.* Montreal, Canada.

Sculley, D., Otey, M., Pohl, M., Spitznagel, B., Hainsworth, J., & Zhou, Y. (2011). Detecting adversarial advertisements in the wild. *Proceedings of the 17th ACM SIGKDD*

*International Conference on Knowledge Discovery and Data Mining.* San Diego, CA, USA.

*Setting up an MLOps environment on Google Cloud*. (s.f.). Obtenido de https://cloud.google.com/architecture/setting-up-an-mlops-environment?hl=es-419

Treveil, M., Omont, N., Stenac, C., Lefevre, K., Phan, D., Zentici, J., . . . Heidmann, L. (2020). *Introducing MLOps.* O'Reilly Media.

*What the Ops are you talking about?* (s.f.). Obtenido de https://towardsdatascience.com/what-the-ops-are-you-talking-about-518b1b1a2694

*Why Is There No DevOps Manifesto?* (s.f.). Obtenido de https://devops.com/no-devops-manifesto/

Zheng, A. (2014). The challenges of building machine learning tools for the masses. *NIPS Workshop.* SE4ML: Software Engineering for Machine Learning.

# 7. Anexo

## 7.1. Tablas comparativa y análisis de las principales herramientas

### 7.1.1. Entorno y Contenerización

| Herramienta | Enlace | Año | Open-Source | Alojado | Gestionado | Tipo de virtualización | Tipo de contenedores |
|---|---|---|---|---|---|---|---|
| **Docker** | https://www.docker.com/ | 2013 | Sí | Local o en la nube (Docker Hub) | Sí (Docker Enterprise) | Contenedores | Docker |
| **Podman** | https://podman.io/ | 2018 | Sí | Local o en la nube (Quay.io, Red Hat Container Catalog) | No | Contenedores | Docker y PodMan |
| **VirtualBox** | https://www.virtualbox.org/ | 2007 | Sí | Local | No | Máquinas virtuales | - |
| **Kubernetes** | https://kubernetes.io/es/ | 2014 | Sí | En la nube (Google Kubernetes Engine, Amazon EKS, Microsoft Azure Kubernetes Service, etc.) o en local | Sí (en la nube a través de servicios gestionados como Google Kubernetes Engine, Amazon EKS, Microsoft Azure Kubernetes Service, etc.) | Contenedores | Docker |
| **Docker Swarm** | https://docs.docker.com/engine/swarm/ | 2015 | Sí | En la nube (Docker Cloud) o en la máquina huésped | Sí (Docker Enterprise) | Contenedores | Docker |
| **Nomad** | https://www.nomadproject.io/ | 2015 | Sí | Local o en la nube (como servicio) | Sí (Nomad Enterprise) | Contenedores | Docker y PodMan |

| Herramienta | Interfaz de usuario | Comunidad y soporte | Compatibilidad con orquestadores | Ventajas frente a competidores | Madurez | Curva de aprendizaje | Precio |
|---|---|---|---|---|---|---|---|
| **Docker** | CLI y GUI | Amplia comunidad y soporte de Docker Inc. | Sí, compatible con Kubernetes y Docker Swarm | Amplia comunidad y soporte de Docker Inc., gran cantidad de imágenes disponibles, facilidad de uso | Madura | Moderada | Gratis, con opciones de pago para Docker Enterprise |
| **Podman** | CLI | Comunidad emergente y soporte de Red Hat | Sí, compatible con Kubernetes | Integración con herramientas de Red Hat, compatibilidad con Docker, no requiere demonio, facilidad de uso | Emergente | Moderada | Gratis |
| **VirtualBox** | GUI | Amplia comunidad y soporte de Oracle | No | Amplia compatibilidad de sistemas operativos huéspedes, buena gestión de redes, facilidad de uso | Madura | Alta | Gratis |
| **Kubernetes** | CLI y GUI | Amplia comunidad y soporte de Kubernetes | Sí | Amplia comunidad y soporte, escalabilidad, compatibilidad con múltiples orquestadores | Madura | Alta | Gratis, con opciones de pago para servicios gestionados |
| **Docker Swarm** | CLI | Comunidad de Docker y soporte de Docker Inc. | Sí | Facilidad de uso, escalabilidad, compatibilidad con múltiples orquestadores | Madura | Moderada | Gratis, con opciones de pago para Docker Enterprise |
| **Nomad** | CLI y GUI | Comunidad emergente y soporte de HashiCorp | Sí | Versatilidad en la orquestación de carga de trabajo, integración con otras herramientas de HashiCorp, escalabilidad | Emergente | Moderada | Gratis, con opciones de pago para Nomad Enterprise |

### 7.1.2. Registro de experimentos

| Herramienta | Enlace | Año | Open-Source | Alojado | Gestionado | Lenguajes compatibles | Integración con herramientas de aprendizaje automático |
|---|---|---|---|---|---|---|---|
| **Mlflow** | https://mlflow.org/ | 2018 | Sí | On-premises o en la nube | Depende de la modalidad de Alojamiento | Python, R, Java, y otras bibliotecas de aprendizaje automático | Integra fácilmente con bibliotecas de aprendizaje automático populares |
| **Neptune** | https://neptune.ai/ | 2018 | No | En la nube | Sí | Python, R, Java, Scala, y otras bibliotecas de aprendizaje automático | Integra fácilmente con bibliotecas de aprendizaje automático populares |
| **Azure Machine Learning Model Registry** | https://azure.microsoft.com/es-es/services/machine-learning/model-management/ | 2019 | No | En la nube | Sí | Python, R, Java, y otras bibliotecas de aprendizaje automático | Integración nativa con Azure Machine Learning Services, así como con herramientas de aprendizaje automático de terceros |
| **Vertex Model Registry** | https://cloud.google.com/vertex-ai/docs/general/model-registry | 2019 | No | En la nube | Sí | TensorFlow, PyTorch, scikit-learn, XGBoost, y otras bibliotecas de aprendizaje automático | Integración nativa con Google Cloud AI Platform, así como con herramientas de aprendizaje automático de terceros |
| **Sagemaker Model Registry** | https://aws.amazon.com/sagemaker/model-registry/ | 2017 | No | En la nube | Sí | TensorFlow, PyTorch, scikit-learn, XGBoost, y otras bibliotecas de aprendizaje automático | Integración nativa con AWS Sagemaker, así como con herramientas de aprendizaje automático de terceros |

| Herramienta | Roles y seguridad | Interfaz de usuario | Comunidad y soporte | Madurez | Curva de aprendizaje | Precio |
|---|---|---|---|---|---|---|
| **Mlflow** | Soporte básico de roles y permisos | CLI y GUI | Buena | Estable | Moderada | Gratis |
| **Neptune** | Sistema avanzado de roles y permisos | CLI y GUI | Buena | Estable | Moderada | Gratis para proyectos individuales y de inicio, planes de pago basados en el uso para empresas |
| **Azure Machine Learning Model Registry** | Sistema avanzado de roles y permisos | CLI y GUI | Buena | Estable | Moderada | Precios basados en el uso y el almacenamiento |
| **Vertex Model Registry** | Sistema avanzado de roles y permisos | CLI y GUI | Buena | Estable | Moderada | Precios basados en el uso y el almacenamiento |
| **Sagemaker Model Registry** | Sistema avanzado de roles y permisos | CLI y GUI | Buena | Estable | Moderada | Precios basados en el uso y el almacenamiento |

### 7.1.3. Orquestación de pipelines

| Herramienta | Enlace | Año | Open-Source | Alojado | Gestionado | Características principales | Interfaz de usuario |
|---|---|---|---|---|---|---|---|
| **Kubeflow** | https://www.kubeflow.org/ | 2017 | Sí | On-premises o en la nube | No | Integración con TensorFlow, PyTorch, XGBoost, entre otros. Escalable a través de Kubernetes | CLI y GUI (a través de servicios adicionales) |
| **Airflow** | https://airflow.apache.org/ | 2015 | Sí | On-premises o en la nube | No | Integración con Hadoop, Spark, Hive, entre otros. Escalable a través de Apache Mesos o Kubernetes | CLI y GUI |
| **Azure Machine Learning Pipelines** | https://azure.microsoft.com/en-us/services/machine-learning/machine-learning-pipelines/ | 2019 | No | En la nube (Azure) | Sí | Integración con herramientas de Azure, como Azure Machine Learning Studio. Escalable a través de Azure Kubernetes Service | CLI y GUI |
| **Vertex Pipelines** | https://cloud.google.com/vertex-ai/docs/pipelines/introduction | 2020 | No | En la nube (Google Cloud) | Sí | Integración con herramientas de Google Cloud, como Vertex AI. Escalable a través de Google Kubernetes Engine. Ofrece automatización y control de versiones | CLI y GUI |
| **Amazon SageMaker Pipelines** | https://aws.amazon.com/sagemaker/pipelines/ | 2020 | No | En la nube (AWS) | Sí | Integración con herramientas de AWS, como SageMaker. Escalable a través de Amazon Elastic Kubernetes Service. Ofrece automatización y control de versiones. Gran seguridad y privacidad | CLI y GUI |

| Herramienta | Lenguaje | Integración de herramientas | Escalabilidad | Comunidad y soporte | Madurez | Curva de aprendizaje | Precio |
|---|---|---|---|---|---|---|---|
| **Kubeflow** | Python | Integración con TensorFlow, PyTorch, XGBoost, entre otros. | Escalable a través de Kubernetes | Activa comunidad de desarrolladores | Madura | Media | Gratuito |
| **Airflow** | Python | Integración con Hadoop, Spark, Hive, entre otros. | Escalable a través de Apache Mesos o Kubernetes | Gran comunidad y soporte activo | Madura | Baja | Gratuito |
| **Azure Machine Learning Pipelines** | Python | Integración con herramientas de Azure, como Azure Machine Learning Studio. | Escalable a través de Azure Kubernetes Service | Soporte de Microsoft y gran comunidad de usuarios | Madura | Media | Precio variable según el uso |
| **Vertex Pipelines** | Python | Integración con herramientas de Google Cloud, como Vertex AI. | Escalable a través de Google Kubernetes Engine | Soporte de Google y gran comunidad de usuarios | Madura | Baja | Precio variable según el uso |
| **Amazon SageMaker Pipelines** | Python | Integración con herramientas de AWS, como SageMaker | Escalable a través de Amazon Elastic Kubernetes Service | Soporte de Amazon y gran comunidad de usuarios | Madura | Baja | Precio variable según el uso |

### 7.1.4. CI/CD

| Herramienta | Enlace | Año | Open-Source | Alojado | Gestionado | Facilidad de configuración y personalización | Compatibilidad con sistemas operativos | Interfaz de usuario |
|---|---|---|---|---|---|---|---|---|
| **Jenkins** | https://www.jenkins.io/ | 2004 | Sí | On-premises o en la nube | Depende de la modalidad de Alojamiento | Alta | Multiplataforma | Web y CLI |
| **Azure Pipelines** | https://azure.microsoft.com/en-us/services/devops/pipelines/ | 2018 | No | En la nube (Azure) | Sí | Alta | Multiplataforma | Web |
| **Cloud Build** | https://cloud.google.com/build | 2018 | No | En la nube (Google Cloud) | Sí | Media | Multiplataforma | Web |
| **AWS CodePipeline** | https://aws.amazon.com/codepipeline/ | 2015 | No | En la nube (AWS) | Sí | Media | Multiplataforma | Web |

| Herramienta | Comunidad y soporte | Madurez | Curva de aprendizaje | Facilidad de uso | Precio |
|---|---|---|---|---|---|
| **Jenkins** | Amplia comunidad y soporte a través de la comunidad y de empresas especializadas | Muy maduro | Alta | Media | Gratuito |
| **Azure Pipelines** | Amplia comunidad y soporte a través de Microsoft y de empresas especializadas | Maduro | Media | Alta | Gratis para proyectos Open Source, planes de pago para proyectos privados |
| **Cloud Build** | Soporte a través de Google y empresas especializadas | En desarrollo | Baja | Alta | Planes de pago según el consumo |
| **AWS CodePipeline** | Amplia comunidad y soporte a través de Amazon y de empresas especializadas | Maduro | Media | Alta | Planes de pago según el consumo |

### 7.1.6. Feature Store

| Herramienta | Enlace | Año | Open-Source | Alojado | Gestionado | Lenguajes compatibles | Almacenamiento de datos | Interfaz de usuario |
|---|---|---|---|---|---|---|---|---|
| **Hopsworks** | https://www.logicalclocks.com/hopsworks/ | 2018 | Sí | On-premises o en la nube (AWS, GCP, Azure) | Depende de la modalidad de alojamiento | Python, Java, Scala | HopsFS | GUI |
| **Azure Databricks** | https://azure.microsoft.com/en-us/services/devops/pipelines/ | 2018 | No | En la nube (Azure) | Sí | Python, R, Scala, SQL | Azure Data Lake Storage, Azure Blob Storage | GUI |
| **Vertex Feature Store** | https://cloud.google.com/vertex-ai/docs/featurestore | 2021 | No | En la nube (Google Cloud) | Sí | Python, Java, Go, C++, Rust, TensorFlow | Google Cloud Storage, BigQuery | GUI |
| **Amazon SageMaker Feature Store** | https://aws.amazon.com/sagemaker/feature-store/ | 2020 | No | En la nube (AWS) | Sí | Python, R, Java, Scala, TensorFlow, PyTorch, MXNet | Amazon S3, Amazon Redshift, Amazon DynamoDB | CLI y GUI |

| Herramienta | Integración con otras herramientas | Madurez | Curva de aprendizaje | Precio |
|---|---|---|---|---|
| **Hopsworks** | Kafka, Spark, TensorFlow, PyTorch | Madura | Moderada | Pago por uso |
| **Azure Databricks** | Azure Synapse Analytics, Power BI, Azure Machine Learning | Madura | Moderada | Pago por uso |
| **Vertex Feature Store** | TensorFlow, Kubeflow, MLflow | Madura | Moderada | Pago por uso |
| **Amazon SageMaker Feature Store** | Amazon SageMaker, AWS Glue, AWS Lambda, AWS Step Functions | Madura | Moderada | Pago por uso |

### 7.1.7. Entrega/Despliegue (Serving)

| Herramienta | Enlace | Año | Open-Source | Alojado | Gestionado | Lenguajes compatibles | Integración con plataformas de nube | Interfaz de usuario |
|---|---|---|---|---|---|---|---|---|
| **Seldon Core** | https://www.seldon.io/ | 2018 | Sí | On-premise, serverless | No | Python, Java, R | Kubernetes, Amazon EKS, Google Kubernetes Engine | CLI |
| **KServe** | https://kserve.github.io/website/ | 2019 | Sí | On-premise, serverless | No | Python, Java, Node.js | Kubernetes | CLI |
| **BentoML** | https://bentoml.org/ | 2020 | Sí | On-premise, serverless | No | Python | Kubernetes, Docker, AWS Lambda, Google Cloud Functions, Azure Functions | CLI y GUI |
| **Azure Machine Learning Endpoints** | https://azure.microsoft.com/en-us/services/machine-learning/endpoints/ | 2018 | No | En la nube (Azure) | Sí | Python, R | Azure | GUI |
| **Vertex Endpoint** | https://cloud.google.com/vertex-ai/docs/endpoints | 2020 | No | En la nube (Google Cloud) | Sí | Python | Google Kubernetes Engine | GUI |
| **SageMaker Endpoint** | https://cloud.google.com/vertex-ai/docs/endpoints | 2017 | No | En la nube (AWS) | Sí | Python | Amazon SageMaker | CLI y GUI |

| Herramienta | Escalabilidad | Despliegue | API | Monitoreo | Ventajas frente a competidores |
|---|---|---|---|---|---|
| **Seldon Core** | Horizontal y vertical | Contenedores Docker | REST, gRPC | Prometheus, Grafana | Amplio soporte para múltiples marcos de aprendizaje automático y capacidad de orquestación avanzada |
| **KServe** | Horizontal y vertical | Contenedores Docker | REST, gRPC | Prometheus, Grafana | Excelente integración con los servicios de Google Cloud, como Kubernetes Engine y AutoML, y capacidad de adaptación a diversas configuraciones de infraestructura |
| **BentoML** | Horizontal y vertical | Contenedores Docker, funciones serverless | REST, gRPC | Prometheus, Grafana | Gran flexibilidad en la construcción y despliegue de modelos de aprendizaje automático, con soporte para múltiples marcos y lenguajes de programación |
| **Azure Machine Learning Endpoints** | Horizontal y vertical | Azure Container Instances, Azure Kubernetes Service | REST | Azure Monitor, Application Insights | Excelente integración con los servicios de Microsoft Azure y una amplia gama de herramientas de aprendizaje automático de código abierto |
| **Vertex Endpoint** | Horizontal y vertical | Google Kubernetes Engine | REST | Cloud Monitoring, Cloud Trace | Fuerte capacidad de automatización de la infraestructura y soporte para la construcción de modelos de aprendizaje automático escalables y de alto rendimiento |
| **SageMaker Endpoint** | Horizontal y vertical | Amazon SageMaker | REST | Amazon CloudWatch | Amplia gama de herramientas y servicios de aprendizaje automático de Microsoft Azure integrados, lo que hace que la creación y despliegue de modelos sea sencilla y escalable |

| Herramienta | Comunidad y soporte | Madurez | Curva de aprendizaje | Precio |
|---|---|---|---|---|
| **Seldon Core** | Activa comunidad y soporte empresarial disponible a través de Seldon Core Enterprise | Estable | Moderada | Gratis |
| **KServe** | Activa comunidad y soporte de Google Cloud | Estable | Moderada | Gratis |
| **BentoML** | Activa comunidad y soporte empresarial disponible a través de BentoML Enterprise | Estable | Moderada | Gratis, versión Enterprise |
| **Azure Machine Learning Endpoints** | Activa comunidad y soporte de Microsoft Azure y documentación completa | Estable | Fácil | Pago por uso |
| **Vertex Endpoint** | Activa comunidad y soporte de Google Cloud y documentación completa | Estable | Fácil | Pago por uso |
| **SageMaker Endpoint** | Activa comunidad y soporte de Amazon Web Services y documentación completa | Estable | Fácil | Pago por uso |