

UNIVERSIDAD DE  
MURCIA



**F** Facultad de  
Informática

TRABAJO FIN DE GRADO

# Machine Learning Operations para el despliegue de modelos de aprendizaje automático

Realizado por  
**María José Bernal Lorca**

Para la obtención del título de  
Grado en Ingeniería Informática

Dirigido por  
Juan Antonio Botía Blaya

Realizado en el departamento de  
Ingeniería de la Información y las Comunicaciones

**Convocatoria de Junio, curso 2021/22**

---

# Resumen

---

En este proyecto estudiaremos MLOps (Machine Learning Operations) como disciplina para mejorar el desarrollo de las distintas fases del ciclo de vida de los proyectos de machine learning. Nos familiarizaremos con sus conceptos clave y descubriremos distintas herramientas que nos permiten automatizar los procesos de nuestros proyectos mediante la aplicación de prácticas MLOps.

Podemos definir MLOps como un conjunto de prácticas que pretenden mejorar la eficacia del despliegue y mantenimiento de los modelos de machine learning. MLOps aparece como una derivación de DevOps, una metodología que tiene como objetivo agilizar el ciclo de vida del software para ofrecer una entrega continua de calidad. En el caso de MLOps, lo que se quiere agilizar y automatizar es el ciclo de vida de los modelos de machine learning.

Una vez que nos hayamos introducido en el concepto de MLOps y conozcamos las herramientas que podemos utilizar para aplicar esta disciplina, nos centraremos en analizar un escenario concreto para el que se quiere diseñar una solución MLOps. Este escenario es el siguiente: los alumnos de la asignatura Aprendizaje Automático de cuarto curso del Grado de Ingeniería Informática desarrollan modelos de machine learning como parte de su trabajo en la asignatura. Cuando un alumno termina el entrenamiento de su modelo, quiere poder publicarlo en un sistema que se encargue de desplegarlo automáticamente. Al mismo tiempo, el profesor de la asignatura quiere poder gestionar los modelos de todos sus alumnos en un mismo lugar, de forma que pueda ejecutarlos y evaluarlos en igualdad de condiciones.

Partiendo de este escenario planteado y de las herramientas estudiadas, se diseñará un sistema MLOps que se encargará de desplegar los modelos creados por los alumnos de forma automática. Esto se conseguirá mediante la integración de distintas herramientas: Seldon Core, Docker, Kubernetes y GitHub Actions. El funcionamiento del sistema es el siguiente: un alumno sube el modelo a un repositorio de GitHub y, gracias a un flujo de trabajo definido con GitHub Actions, el modelo se desplegará en un clúster de Kubernetes que previamente habremos configurado con todos los elementos necesarios para la gestión de los modelos. Después de su despliegue, los modelos quedarán disponibles a través de un API REST, que podrá ser utilizado por el profesor para evaluar los modelos.

Por último, se mostrará el funcionamiento del sistema MLOps mediante el despliegue de un modelo predictivo real.

**Palabras clave:** Machine Learning, MLOps, DevOps, Despliegue, Servicio REST.

---

# Extended Abstract

---

In this project we will present MLOps (Machine Learning Operations) as a discipline aimed to improve the different processes in machine learning lifecycle. We will get acquainted with the MLOps concept and discover different tools that can help us automate our project workflow.

Machine Learning Operations was born as an extension of DevOps methodology. To understand the different aspects of MLOps, we have to know first what is DevOps and what are its peculiarities as a software engineering methodology.

The term DevOps comes from the combination of two words: development (Dev) and operations (Ops). We could define DevOps as a set of practices that intend to unite software development with IT operations.

DevOps main goal is to streamline the software development lifecycle to provide high-quality continuous delivery of software products at a lower cost. DevOps practices are characterized by seeking automation and monitoring of all phases of the software lifecycle. Two key concepts of this methodology are continuous integration (CI) and continuous delivery (CD).

On the one hand, continuous integration is a Software Engineering practice that consists of automating the entire compilation and test execution of a project in order to be able to repeat this process as often as possible. By repeating the building and tests running process continuously, bugs are caught faster, so we can avoid dragging bugs into the next phases of the life cycle.

On the other hand, continuous delivery is a practice that proposes a faster, more frequent and reproducible building, testing and release of software. This practice proposes to create incremental versions of the software so we make sure we are not adding too many major changes to the software at once. This helps reduce the cost, time, and risk of software release.

In this context, Machine Learning Operations ( MLOps) appears as a derivation of DevOps. MLOps seeks to increase automation and improve the quality of production models, having continuous integration and continuous delivery as key elements.

Once we have been introduced to the key concepts of MLOps, let's explore tools that can be used to carry out our ML projects with a MLOps perspective. We will present platforms that offer different tools to cover the automation of the entire project lifecycle. Kubeflow, MLflow and Metaflow are considered complete MLOps platforms. We will also present tools that focus on model deployment and monitoring. Some of these tools are Seldon Core, KServe and BentoML. Finally, we will present tools that are not specifically from the field of machine learning, but can be very useful when deploying applications in general. These tools are Docker, Kubernetes and Github Actions.

After discovering all these tools, we will analyze a scenario where MLOps is needed. The scenario is as follows: Computer Science students are developing machine learning models as part of their work in the subject “Aprendizaje Automático”. When students finish training their models, they want to publish them in a system that will automatically deploy the models for them. At the same time, the professor of this subject wants to have a place where he can manage and test all the models deployed by the students. In this place, he will be able to run and evaluate the models in equal conditions.

Starting from this scenario and the tools we have studied, we will design a MLOps system that will be in charge of automatically deploying machine learning models created by the students. This will be achieved thanks to the integration of different tools: Seldon Core, Docker, Kubernetes and GitHub Actions.

Seldon Core offers solutions aimed to deploy different types of ML projects. Our main goal is to deploy machine learning models based on predictive methods in R that use specific libraries from this language. The best option that Seldon Core can bring in this case are the language wrappers.

Language wrappers are, summing up, a little layer of code that has the function of translating an existing software interface (in a specific language) to increase its compatibility with other software elements. In this case, we want to wrap a model in R language to make it compatible with the deployment process of Seldon Core.

To wrap our models in a specific language, like R or Python, Seldon Core proposes to use a tool called Source-to-Image (S2I), by OpenShift. This tool allows us to create images that can be deployed in Docker containers from source code. Starting from a building image which, in our case, will be the default image for R Models by Seldon Core, we will be able to build the image of our model ready for deployment.

Seldon Core also provides an standardized OpenAPI/Swagger user interface to send requests to the models deployed in Kubernetes cluster.

Kubernetes is going to be a big part of our MLOps solution, so it's important that we know some key aspects of this tool. On one hand, we have Kubernetes objects, among which are Kubernetes namespaces and pods. Namespaces act like virtual clusters inside the real cluster. Namespaces provide a scope for names. Names of resources need to be unique within a namespace, but not across namespaces. A pod is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers. In our case, each model image will be deployed in an exclusive pod.

On the other hand, we have Kubernetes nodes, which are virtual machines that are in charge of running the containerized applications. In our cluster, we will work with a single node, but if we want to deploy models on a large scale, it would be convenient to have different nodes that contain different instances of the same model, so if one of them fails, we can lean on the other.

Another key point to learn is the concept of a Kubernetes operator. Summing

up, an operator is a software piece that contains the knowledge to act in a specific situation. In our case, we will have to install the Seldon Core operator for Kubernetes. This operator will be responsible for making production, monitoring and maintenance of models easier.

We will also install a service mesh in our cluster. A service mesh is a piece of software that controls how different parts of an application share data with one another. This service is especially necessary when our application follows a microservice based architecture. All the microservices should follow the application's communication rules, so each component has to be configured individually with these rules. The service mesh is, after all, a layer of the infrastructure that contains all the rules that define the communication between the microservices, so that we don't have to configure each of the services manually.

In our scenario, each model will be deployed as a microservice in the Kubernetes cluster. Even if we do not manage too many models at the start, we have to be sure our system is able to scalate easily in case we decide to increase the number of models hosted in the future.

In Kubernetes, deployments are made by using a JSON or YAML file. In this file we will have to indicate the resources we want our model container to have. Among these resources we can indicate, for example, the processing power and the amount of memory used. Seldon Core operator enlarge this aspect of Kubernetes with a customized type of resource, which is called SeldonDeployment. This type of resource allow us to define a runtime inference graph made up of models.

All we have seen so far is the manual process to create a model image in order to deploy it in a Kubernetes cluster. We can complete this process by running order manually: s2i to create the model image, kubectl to manage Kubernetes cluster, docker to manage created model images. . . At this point, we find a new challenge: to automate all this process in a way that, when a new model is generated, it will be packaged and deployed in our cluster automatically. For this automation task we have selected the tool GitHub Actions

GitHub Actions is a tool for continuous integration and continuous delivery of software, which in our case are machine learning models. One of the biggest advantages of this tool is its integration with the famous web repository, GitHub. This tool will be familiar for most students. Even if they have never heard about GitHub Actions directly, they have probably used GitHub repositories before.

Now that we know which tools will be part of our system, let's focus on how the whole process will work. To be able to explain this process, we will divide it in three different stages: uploading the model to the repository, running the actions (create the model image) and deploying the model image in Kubernetes cluster.

On the first stage we will configure our GitHub repository. To define actions with GitHub Actions we will have to create a directory called **.github/workflows** in the root directory of our repository. Inside this directory we will define a workflow that

will run a series of jobs when an event is detected on the repository. These workflows are defined via a YAML file that will contain the actions to run.

After creating and training their models, the students will have to create a directory to host the source code needed to run the machine learning model. The content of this directory has to follow a specific pattern so S2I can build the image properly. Depending on the programming language used, the requirements of this directory will be slightly different. In general, we will have to include a class that shows the logic of our model, a file that contains our model's dependencies and a directory called **.s2i** that contains an environment file where we will indicate some parameters needed by S2I to create the model image.

When a student makes a push action on the repository to include the directory with the model and the dependencies, the workflow defined in **.github/workflows** will be run by GitHub automatically.

GitHub Actions will create a virtual server to run all the steps of the workflow. This workflow will include the following steps:

- Install S2I tool with a predefined action made by OpenShift.
- Run the specific S2I command to create the image of our model. S2I will obtain all the resources needed to create this image from the repository.
- Upload the new model image to DockerHub by using a predefined action. Before executing this step, we will have to configure the users and password of the DockerHub repository as a GitHub encrypted secret.
- Connect to the server that hosts Kubernetes cluster in order to start the deployment.

Then, the process will be in charge of the Kubernetes cluster server. The first step is to pull the model image from DockerHub repository. Once we have the image locally, we will use a YAML file to define the characteristics of our Kubernetes deployment. After that, we should be able to deploy the model by running **kubectl apply -f fichero-seldon-deployment.yml**. All these steps will be run automatically thanks to the definition of a simple script with all the executable command orders.

After all the steps, the model will be accessible through an URL endpoint. We can send petitions to the model through this URL.

Finally, we will show the MLOps system operation by the deployment of a real predictive machine learning model. We will explain all the steps followed to start the system and we will present the deployment test results. In this case, we will focus on the deployment of a machine learning model in R.

The model used for testing the MLOps system is a predictive model to detect diabetes. This model was trained with a dataset from the library `mlbench` that contains the results of medical observations carried out on Pima indigenous tribe population.

**Keywords:** Machine Learning, MLOps, DevOps, Deployment, REST Service.

---

# Índice general

---

<b>1. Introducción</b>	<b>1</b>
1.1. Marco del proyecto . . . . .	1
1.2. Motivación . . . . .	1
1.3. Objetivos . . . . .	2
1.4. Estructura del documento . . . . .	3
<b>2. Estado del Arte</b>	<b>4</b>
2.1. Introducción . . . . .	4
2.2. Machine Learning Operations . . . . .	4
2.2.1. Introducción a DevOps . . . . .	4
2.2.2. ¿Qué es MLOps? . . . . .	5
2.2.3. Ciclo de vida de los modelos de machine learning . . . . .	5
2.3. Herramientas MLOps . . . . .	7
2.3.1. Plataformas MLOps completas . . . . .	8
2.3.2. Desarrollo, despliegue y monitorización . . . . .	9
2.3.3. Herramientas complementarias . . . . .	10
2.4. Conclusiones . . . . .	12
<b>3. Objetivos y metodología</b>	<b>13</b>
3.1. Introducción . . . . .	13
3.2. Definición de objetivos . . . . .	13
3.3. Enfoque del análisis . . . . .	14
3.4. Enfoque del diseño . . . . .	14
3.5. Enfoque del desarrollo . . . . .	14
<b>4. Análisis</b>	<b>15</b>
4.1. Introducción . . . . .	15
4.2. Descripción del escenario . . . . .	15
4.3. Usuarios del sistema . . . . .	16
4.4. Casos de Uso . . . . .	16
4.4.1. Administrador . . . . .	16
4.4.2. Alumno . . . . .	17
4.4.3. Consumidor de modelos . . . . .	17
<b>5. Diseño y desarrollo</b>	<b>18</b>
5.1. Introducción . . . . .	18
5.2. Selección de herramientas . . . . .	18
5.2.1. Seldon Core . . . . .	18
5.2.2. Kubernetes . . . . .	21
5.2.3. GitHub Actions . . . . .	23
5.3. Funcionamiento del sistema . . . . .	24
5.3.1. Fase I. Un modelo se sube al repositorio. . . . .	25



5.3.2.	Fase II. Ejecución de trabajos con GitHub Actions. . . . .	25
5.3.3.	Fase III. Despliegue en Kubernetes. . . . .	25
5.4.	Pruebas realizadas . . . . .	26
5.4.1.	Preparación del clúster . . . . .	26
5.4.2.	Preparación del repositorio y del modelo . . . . .	27
5.4.3.	Creación del flujo de trabajo con GitHub Actions . . . . .	29
5.4.4.	Peticiones a la API del modelo . . . . .	31
<b>6.</b>	<b>Conclusiones y vías futuras</b>	<b>33</b>
<b>7.</b>	<b>Bibliografía</b>	<b>35</b>
<b>A.</b>	<b>Anexo. Manual del administrador</b>	<b>37</b>
A.1.	Instalación de herramientas . . . . .	37
A.1.1.	Docker . . . . .	37
A.1.2.	Kubectrl . . . . .	37
A.1.3.	Minikube . . . . .	38
A.1.4.	Istio . . . . .	38
A.1.5.	Seldon Core . . . . .	39
A.2.	Dashboard de Kubernetes . . . . .	40

---

# Índice de figuras

---

2.1. Algoritmo de machine learning . . . . .	6
2.2. Ciclo de vida de un proyecto ML. Fuente: <a href="https://subscription.packtpub.com">subscription.packtpub.com</a> . . . . .	7
5.1. Interfaz estandarizada para enviar peticiones a modelos. Fuente: <a href="https://docs.seldon.io">docs.seldon.io</a> . . . . .	20
5.2. Clúster de Kubernetes. Fuente: <a href="https://www.theserverside.com">www.theserverside.com</a> . . . . .	21
5.3. Representación gráfica de una malla de servicio. Fuente: <a href="https://www.teldat.com">www.teldat.com</a> . . . . .	22
5.4. GitHub Actions - Ejemplo de fichero YALM . . . . .	23
5.5. Infraestructura del sistema MLOps . . . . .	24
5.6. Formato del directorio . . . . .	28
5.7. Petición del diagnóstico del individuo 1 mediante la interfaz Swagger . . . . .	32
5.8. Respuesta a la petición de diagnóstico del individuo 1 a través de la interfaz Swagger . . . . .	32
5.9. Petición de diagnóstico del individuo 2 y respuesta a través de Curl . . . . .	32

---

# Índice de extractos de código

---

5.1.	Fichero <b>simple.script</b> . . . . .	27
5.2.	Fichero <b>deploy.yml</b> . . . . .	27
5.3.	Fichero <b>MyModel.R</b> . . . . .	28
5.4.	Fichero <b>install.R</b> . . . . .	29
5.5.	Fichero <b>environment</b> . . . . .	29
5.6.	Fichero <b>cicd.yml</b> . . . . .	30

---

# 1. Introducción

---

## 1.1. Marco del proyecto

En este Trabajo Fin de Grado (TFG) se explorará MLOps como disciplina destinada a facilitar la puesta en producción de los modelos de machine learning (ver sección 2.2). Recorreremos las distintas herramientas que aplican este enfoque con el objetivo de conocer el estado actual de las tecnologías (ver sección 2.3). Tras este estudio, nos centraremos en analizar un escenario concreto sobre el que nos interesa aplicar las prácticas propuestas por MLOps (ver capítulo 4). El objetivo final es diseñar una solución que integre las herramientas necesarias para el despliegue de modelos de machine learning de una forma sencilla y automatizada (ver capítulo 5).

A continuación se explicará el escenario del que surge este proyecto. Los alumnos del Grado de Ingeniería Informática, concretamente de la asignatura Aprendizaje Automático de cuarto curso, desarrollan modelos de machine learning como parte de su trabajo en la asignatura. Cuando un alumno termina el entrenamiento de su modelo, quiere poder publicarlo en un sistema que se encargue de desplegarlo automáticamente, consiguiendo que quede accesible para que otros usuarios lo usen. Los alumnos de la asignatura podrán utilizar este sistema para desplegar sus modelos de machine learning, que podrán estar desarrollado tanto en Python como en R. De esta forma, todos los modelos se gestionarán en un mismo lugar, permitiendo que el profesor de la asignatura pueda evaluarlos en igualdad de condiciones.

En resumen, se busca crear un sistema que permita publicar modelos de machine learning de forma centralizada para que puedan utilizarse a modo de producción. Cuando un alumno genere un modelo, podrá subirlo a un repositorio que se encargará de realizar el despliegue automáticamente, haciendo que su API quede disponible como un servicio web de tipo REST.

## 1.2. Motivación

En la actualidad, los científicos de datos dedican gran parte de su tiempo y esfuerzo en crear modelos de aprendizaje automático destinados a infinidad de tareas. Sin embargo, muchos de esos modelos nunca llegan a estar disponibles en entornos de producción. Esto se debe a que, más allá de las barreras técnicas existentes durante el desarrollo de los modelos de machine learning, existen también barreras organizacionales que dificultan la puesta en producción de los modelos.

En este contexto aparece **Machine Learning Operation**, o MLOps, con el objetivo de eliminar la brecha entre la ciencia de datos y la puesta en producción de

los productos software. Podemos definir MLOps como un conjunto de prácticas que tienen como objetivo desarrollar, desplegar y mantener los modelos de aprendizaje automático en entornos de producción. Estas prácticas nacen de la combinación de los elementos más esenciales de la cultura DevOps en Ingeniería del Software, como la integración continua y la distribución continua, con los procesos propios de la ciencia de datos.

La gestión y mantenimiento de los modelos de machine learning no es una tarea precisamente sencilla. En un entorno en el que se producen y mejoran los modelos constantemente, tenemos que asegurarnos de que contamos con una infraestructura que nos permita actualizar los modelos y llevar un control de las versiones de forma satisfactoria.

Como veremos en la sección 2.2.3, los modelos de machine learning hacen predicciones mediante la búsqueda de patrones en los datos. Sin embargo, cuando los datos utilizados se vuelven obsoletos y dejan de reflejar el estado del mundo, ocurre un problema llamado "drift" o desvío de datos. Los modelos que fueron construidos a partir de estos datos deben ser entrenados y desplegados de nuevo para que sus predicciones conserven la calidad. En este contexto, MLOps propone técnicas para la monitorización de modelos y *datasets*, de forma que, en caso de detectarse un problema, se vuelva a iniciar el proceso de entrenamiento y despliegue de los modelos de forma automática. [11]

## 1.3. Objetivos

Este TFG busca explorar MLOps como disciplina para gestionar el ciclo de vida de modelos de machine learning. Para ello, estudiaremos el concepto de MLOps y su importancia en el desarrollo de proyectos de ciencias de datos. Después, profundizaremos en distintas herramientas software que aplican este enfoque para llevar a cabo las distintas fases del ciclo de vida de los modelos. El objetivo es exponer el estado actual de las tecnologías, destacando las ventajas que ofrecen y qué fases del ciclo de vida se pretenden abarcar.

Tras investigar el estado actual de las tecnologías, nos centraremos en un escenario concreto sobre el que queremos aplicar el enfoque MLOps. Este escenario, como se ha expuesto en la primera sección de este capítulo, consiste en que los alumnos quieren poder desplegar sus modelos de machine learning de forma centralizada para que otros usuarios, especialmente el profesor de la asignatura, puedan probarlos y evaluarlos. Analizaremos las necesidades de los usuarios y las posibles soluciones MLOps que encajan en este contexto.

El objetivo final es presentar una solución que integre los conocimientos y herramientas estudiados para diseñar un sistema en el que los alumnos puedan publicar sus modelos de forma centralizada. La idea final es presentar una solución que permita desplegar modelos automáticamente, de forma que puedan recibir peticiones a través de una API de tipo REST. Por otro lado, el profesor podrá acceder a esta aplicación

web para realizar peticiones a los distintos modelos.

## 1.4. Estructura del documento

A continuación vamos a presentar los distintos capítulos que componen este documento junto a un breve resumen de su contenido:

- En el Capítulo 2 se realizará un análisis del estado del arte de MLOps. Se presentarán los conceptos clave de este conjunto de prácticas, así como sus tecnologías asociadas.
- En el Capítulo 3 se determinarán los objetivos de este proyecto más allá de explorar MLOps y se explicarán los enfoques utilizados para alcanzarlos.
- En el Capítulo 4 analizaremos el caso de estudio del que parte este TFG. Se estudiarán las características del escenario y de los usuarios que en él se presentan.
- En el Capítulo 5 se presentará la infraestructura de nuestro sistema MLOps. Se profundizará en las herramientas que la componen, se explicará el proceso de desarrollo del sistema y se mostrarán las pruebas de despliegue realizadas.
- En el Capítulo 6 veremos las conclusiones que se han sacado durante el desarrollo de este proyecto, así como las mejoras que podrían aplicarse en un futuro.

---

## 2. Estado del Arte

---

### 2.1. Introducción

En este capítulo realizaremos una introducción a los conceptos más relevantes de MLOps y su importancia durante el desarrollo del ciclo de vida de los modelos de machine learning. También presentaremos herramientas que siguen esta disciplina para desplegar sistemas de aprendizaje automático de una forma eficaz y eficiente.

El objetivo de este capítulo es obtener una visión general del estado actual de las tecnologías y de los procesos referentes a la automatización del desarrollo y puesta en producción de los modelos. Presentaremos y se analizarán las distintas herramientas software, destacando su interés en el marco de desarrollo de este trabajo.

### 2.2. Machine Learning Operations

#### 2.2.1. Introducción a DevOps

Machine Learning Operations nace como una extensión de la metodología DevOps, por lo que para entender algunos de los conceptos más importantes de MLOps primero tenemos que estar familiarizados con el concepto de DevOps y sus características.

El término de DevOps proviene de una combinación de los términos "development" (Dev), que en español significa desarrollo, y "operations" (Ops), que significa operaciones. En definitiva, podríamos definir DevOps como un conjunto de prácticas que pretenden agrupar el desarrollo del software y las operaciones propias de las tecnologías de la información (TI).

El objetivo principal de DevOps es agilizar el ciclo de vida del desarrollo de software para poder proporcionar una entrega continua de alta calidad a un menor coste. Las prácticas propuestas por DevOps se caracterizan por defender la automatización y monitorización de todas las fases del ciclo de vida del software. Dos de los conceptos clave de DevOps son la integración continua (CI) y la entrega continua (CD). [20]

Por un lado, la integración continua es una práctica de Ingeniería del Software que consiste en la automatización de todo el proceso de compilación y ejecución de pruebas de un proyecto con la finalidad de poder repetirlo con la mayor frecuencia posible. Al repetir el proceso de compilación y ejecución de pruebas continuamente, los fallos se detectan más rápidamente y se evita arrastrar errores a las siguientes fases del ciclo de vida. [19]

Por otro lado, la entrega continua es una práctica que propone una construcción, prueba y liberación del software más rápida, frecuente y repetible. Propone crear versiones incrementales, realizando entregas continuas sin necesariamente haber añadido grandes cambios al software. De esta forma se reduce el costo, el tiempo y el riesgo de la liberación del software. [18]

### 2.2.2. ¿Qué es MLOps?

Machine Learning Operations, o MLOps, hace referencia a un conjunto de prácticas que pretenden mejorar la eficacia y calidad del despliegue y mantenimiento de modelos de aprendizaje automático. MLOps aparece como una derivación de DevOps, una metodología que, como hemos visto, tiene como objetivo agilizar el ciclo de vida del desarrollo de productos software para ofrecer una entrega continua de calidad. En el caso de MLOps, lo que se quiere agilizar y automatizar es el ciclo de vida de los modelos de aprendizaje automático, basándose, al igual que DevOps, en el concepto de integración continua y distribución continua (CI/CD). [22]

A diferencia de DevOps, en MLOps la integración continua no consiste únicamente en probar y validar el código fuente, sino también en validar datos, esquemas y modelos. Por otro lado, la distribución continua no consiste en la entrega de un solo paquete de software, sino en la entrega de un sistema formado por una canalización o pipeline de entrenamiento que debe implementar un servicio de predicción o inferencia del modelo de manera automática.

La implementación de las prácticas MLOps se está convirtiendo en un componente crítico del desarrollo exitoso de los proyectos de ciencia de datos, tanto a nivel empresarial como a nivel investigativo. A pesar de ser un concepto bastante nuevo, la aplicación de MLOps ha demostrado ser una gran ayuda para conseguir que los proyectos generen valor a largo plazo, además de reducir el riesgo asociado a las iniciativas de ciencia de datos, aprendizaje automático e inteligencia artificial.

### 2.2.3. Ciclo de vida de los modelos de machine learning

Antes de presentar en qué consiste el ciclo de vida de los modelos, tenemos que tener claro algo mucho más básico: ¿qué es exactamente un modelo de machine learning?

Un modelo de machine learning es, en definitiva, un programa que se ha generado mediante el entrenamiento con datos de un algoritmo de machine learning. Podemos ver este programa como una caja negra que es capaz de procesar una entrada de datos para proporcionar una salida que vendrá determinada por el entrenamiento recibido. Los modelos aceptan llamadas a su función "predict", encargada de generar una predicción a partir de los datos de entrada. Por ejemplo, cuando un modelo predictivo recibe unos datos, es capaz de generar un pronóstico a partir de ellos basándose en los datos que se usaron para su entrenamiento. En la Figura 2.1 podemos ver un esquema que representa



el proceso de entrenamiento de un algoritmo de machine learning. Concretamente, un algoritmo que genera un modelo en forma de una base de reglas de decisión. Toma como entrada ejemplos de lo que debería generar el modelo como “Respuesta” para cada entrada de “Datos”. Se suelen usar miles, cientos de miles de ejemplos concretos para que el algoritmo sea capaz de generar dicho modelo.

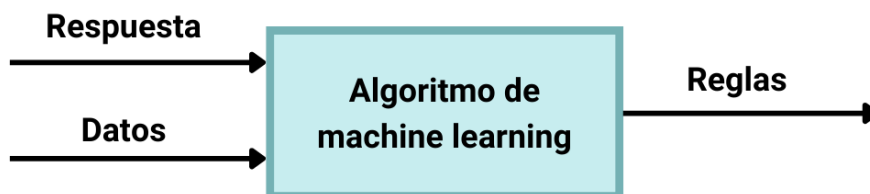


Figura 2.1: Algoritmo de machine learning

EL enfoque de Machine Learning Operations puede aplicarse en una, varias o todas las fases del ciclo de vida de los proyectos de machine learning. Para entender mejor la importancia de este enfoque, recorreremos las distintas fases. En la Figura 2.2 podemos ver un diagrama del ciclo de vida de los proyectos.

Para construir un modelo de machine learning, el primer paso es seleccionar y revisar los datos que queremos utilizar, así como controlar los formatos en los que se presentan estos datos. En esta fase de manejo de datos, MLOps propone realizar un versionado de los datos y de sus atributos, de forma que se mantenga un control de los conjuntos de datos que han sido utilizados durante la construcción del modelo. En la Figura 2.2 esta fase corresponde a las casillas de color amarillo, bajo el nombre de “Data Retrieval” y “Data Preparation”.

El siguiente paso en el flujo de trabajo de machine learning es la propia construcción del modelo, también conocida como fase de entrenamiento. En general, esta fase de experimentación supone realizar distintas pruebas hasta encontrar la combinación que genera el modelo que buscamos. En este caso, MLOps propone llevar un control de las pruebas realizadas haciendo un seguimiento de las métricas de los experimentos y controlando la fuente del código y las dependencias. Por otro lado, propone aplicar el concepto de pipeline al ciclo de vida de los modelos. Esta idea consiste en crear una secuencia de tareas propias del flujo de trabajo para poder realizar checkpoints en cada paso y repetir el proceso cuando se realice algún cambio. En la Figura 2.2 esta fase corresponde a las casillas de color azul, bajo el nombre de “Modeling” y “Model Evaluation and Tuning”.

Una vez que el modelo ha sido creado, pasamos a la fase de validación. Esta fase consiste en la preparación del modelo para la puesta en producción. Incluye el despliegue de los modelos en entornos de pruebas controlados para comprobar el funcionamiento de los modelos. Esta validación también suele necesitar un despliegue por etapas, por lo que en este caso MLOps también propone técnicas para la automatización. En la Figura 2.2 no aparece esta fase como tal, ya que todo el proceso de despliegue (incluyendo pruebas de despliegue) se engloba en la casilla de color rosa

bajo el nombre "Deployment & Monitoring".

Tras las pruebas de validación, el modelo podrá ser desplegado en un entorno de producción real. En este punto, MLOps propone monitorizar el comportamiento de los modelos. Se controlarán métricas técnicas, como por ejemplo, el rendimiento y la latencia, así como métricas propias de los modelos, como la desviación del modelo o la desviación de datos. Al aparecer estos desvíos, los modelos deberán volver a ser entrenados. MLOps propone monitorizar los modelos para detectar desviaciones y poder iniciar de nuevo el proceso de entrenamiento de forma automática. Como hemos dicho, en la Figura 2.2 esta fase corresponde a la casillas de color rosa bajo el nombre "Deployment & Monitoring".

En general, la idea se resumen en crear pipelines con una sucesión de tareas que pueda automatizarse para que, en caso de detectarse algún problema, se pueda repetir todo el proceso de entrenamiento, validación y despliegue de una forma eficiente y eficaz, garantizando la calidad de los modelos de aprendizaje automático.

En este trabajo profundizaremos especialmente en la fase de despliegue, ya que el escenario del que parte este TFG requiere desplegar modelos a modo de servicio, de forma que puedan recibir peticiones y generar respuestas.

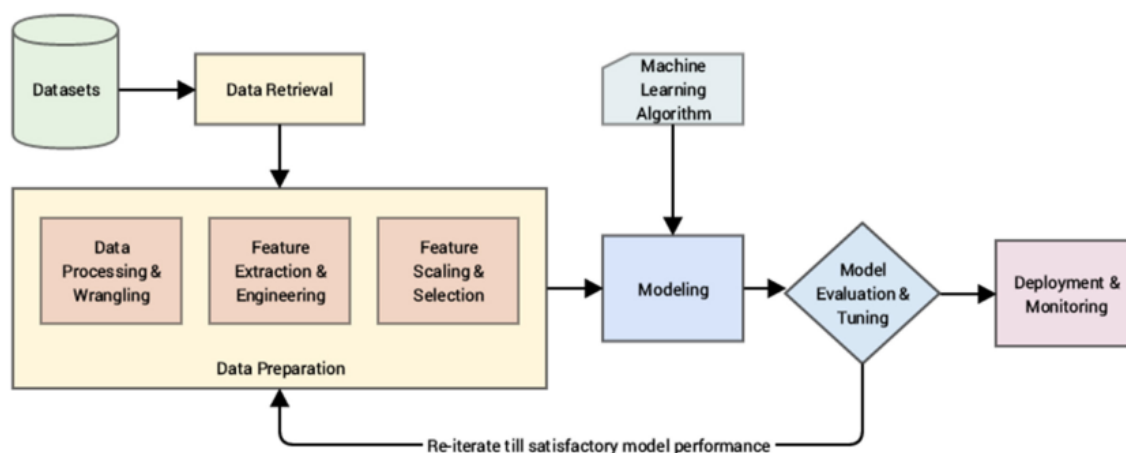


Figura 2.2: Ciclo de vida de un proyecto ML. Fuente: [subscription.packtpub.com](https://subscription.packtpub.com)

## 2.3. Herramientas MLOps

En esta sección se presentarán distintas herramientas open-source que tienen como objetivo aplicar el enfoque MLOps para garantizar el desarrollo satisfactorio de todo el ciclo de vida de los modelos de machine learning, especialmente aquellas que puedan ser de utilidad en el escenario de este TFG. Las herramientas se presentarán organizadas según su cometido, teniendo en cuenta en qué porción concreta del ciclo

de vida del modelo de ML se centran.

### 2.3.1. Plataformas MLOps completas

En esta sección presentaremos algunas plataformas que contienen herramientas para gestionar todo el flujo de vida de los modelos de machine learning. Aunque en general estas plataformas ofrecen herramientas de trabajo para organizaciones o grupos grandes de desarrolladores, nos viene bien conocerlas y saber qué ofrecen.

#### Kubeflow

Kubeflow es una plataforma de aprendizaje automática gratuita y de código abierto dedicada a la creación de pipelines para orquestar y ejecutar flujos de trabajo de proyectos ML en clústeres de Kubernetes (ver apartado [2.3.3](#)).

Esta plataforma comenzó como un proyecto interno de Google para ejecutar trabajos de TensorFlow, una biblioteca para aprendizaje automático (también de Google), de una forma más sencilla. Actualmente ofrece integración con otras herramientas open-source para servir modelos como Seldon Core o MLRun. [\[1\]](#) [\[21\]](#)

En conclusión, Kubeflow se presenta como una plataforma enfocada a satisfacer los requisitos de equipos grandes de producción de soluciones de machine learning. Sin embargo, en el escenario del que partimos, esta plataforma queda un poco grande, ya que esta destinada a equipos grande que quieren automatizar todo el ciclo de vida de los modelos. Además, esta plataforma requiere de especialistas que lleven un mantenimiento de la infraestructura de Kubernetes.

#### MLflow

MLflow también es una plataforma de código abierto destinada a la administración de todo el ciclo de vida de las soluciones ML. MLflow fue creada por Databricks y ofrece distintas funciones, como el seguimiento, la gestión y el empaquetado de modelos, así como un repositorio centralizado de modelos. [\[14\]](#)

En general, MLflow es más fácil de configurar que Kubeflow, ya que no requiere conocer la infraestructura de Kubernetes. Podríamos considerar que MLflow está más enfocada a satisfacer las necesidades de los científicos de datos que buscan organizar mejor sus experimentos y sus modelos de machine learning.

Sin embargo, al igual que Kubeflow, esta herramienta sigue estando enfocada a la automatización del ciclo completo de proyectos de machine learning. En nuestro caso buscamos una herramienta que nos permita automatizar el despliegue sin complicaciones de por medio.

## Metaflow

Metaflow es una plataforma MLOps de código abierto desarrollada inicialmente por Netflix. Es una herramienta escrita en Python/R que facilita la creación y gestión de proyectos empresariales de ciencia de datos.

Metaflow integra bibliotecas de aprendizaje automático, aprendizaje profundo y big data basadas en Python para entrenar, implementar y administrar modelos de aprendizaje automático de manera eficiente. [12]

Esta herramienta se centra en abarcar la fase de experimentación del flujo de trabajo de los proyecto ML en un ámbito empresarial, por lo que no nos interesa demasiado para la solución MLOps que queremos desarrollar en este TFG.

### 2.3.2. Desarrollo, despliegue y monitorización

En esta sección presentaremos algunas herramientas que están especialmente centradas en las fases de desarrollo, despliegue y monitorización, especialmente en estas dos últimas.

## CML

Continuous Machine Learning (CML) es una herramienta CLI de código abierto que aplica un enfoque MLOps para implementar la integración continua y la entrega continua. Esta herramienta está centrada en la automatización del desarrollo de los modelos, proporcionando aprovisionamiento de máquinas virtuales para las ejecuciones, comparación de experimentos y monitorización de conjuntos de datos, así como entrenamiento y evaluación de modelos. [7]

Esta herramienta es interesante para los científicos de datos que producen los modelos, ya que se centra principalmente en la fase de desarrollo. Sin embargo, como la tarea principal a abordar en este TFG es el despliegue de modelos, esta herramienta no nos será de mucha utilidad.

## Seldon Core

Seldon Core es una plataforma de código abierto para el despliegue de modelos de machine learning en Kubernetes. Es compatible con numerosas bibliotecas de aprendizaje automático y contenedores de lenguaje, permitiendo convertir cualquier modelo en un microservicio REST en producción.

Esta herramienta puede integrarse con otros entornos y bibliotecas específicas, como Tensorflow o Pytorch, para desplegar los modelos. Además también ofrece la opción de utilizar wrappers de lenguaje para empaquetar nuestros modelos antes del

despliegue. Esta última opción nos resulta de especial interés, ya que nos permitiría desplegar modelos en distintos lenguajes de programación, como R o Python. [10]

En general, Seldon Core parece ofrecer todo lo que buscamos. Se centra principalmente en el despliegue de los modelos. Además, también ofrece opciones para monitorizar los modelos unas vez desplegados.

## **KServe**

KServe es una herramienta que, al igual que Seldon Core, también se basa en el uso de un clúster de Kubernetes. Esta herramienta está destinada a servir modelos de machine learning en distintos frameworks. Proporciona interfaces que pueden integrarse con algunas de las plataformas de machine learning más comunes, como Tensorflow, XGBoost, ScikitLearn, PyTorch y ONNX. [8]

Esta herramienta está destinada al despliegue y monitorización de modelos, por lo que, en principio, podría ser de interés para el desarrollo de este trabajo. El único inconveniente es que KServe está pensada para trabajar junto a las plataformas ML que hemos nombrado arriba. Para nuestro escenario buscamos una solución más directa, que permita desplegar modelos en varios lenguajes sin depender de otras plataformas o ecosistemas.

## **BentoML**

BentoML es un framework unificado de código abierto que pretende simplificar el despliegue de modelos para ponerlos en producción de forma sencilla. Esta herramienta se presenta como una alternativa más simple y rápida para la entrega continua de modelos. Puede integrarse con la mayoría de frameworks para entrenamiento de modelos, como MLflow, ONNX, PyTorch, etc. [2]

Al igual que Seldon Core, BentoML trabaja con Docker y Kubernetes y podría ofrecernos la funcionalidad que buscamos, ya que permite servir modelos rápidamente a través de un endpoint de tipo API HTTP.

### **2.3.3. Herramientas complementarias**

En este apartado se presentarán herramientas que, aunque no son exclusivas del ámbito de la ciencia de datos y de MLOps, son de gran utilidad desde el punto de vista de la automatización del despliegue del software con un enfoque de integración continua y entrega continua. Muchos de los frameworks y herramientas MLOps trabajan junto a las soluciones software que trataremos en este apartado para lograr sus objetivos a la hora de poner los modelos en producción.

## Docker

Docker es una plataforma de código abierto que permite crear y construir aplicaciones dentro de contenedores. Estos contenedores de software suponen una forma de virtualización del sistema operativo. Podríamos decir que son paquetes que contienen todos los recursos necesarios para ejecutar una aplicación software: ejecutables, código fuente, bibliotecas, archivos de configuración, etc. Todo esto nos permite crear aplicaciones portables, que requieren menos recursos del sistema y que facilitan la integración continua. [6]

Desde el punto de vista de la ciencia de datos, Docker nos permite crear imágenes de nuestros modelos, de manera que pueden ser encapsulados y ejecutados en contenedores. De esta forma podemos encapsular un entorno con el modelo y con todos los recursos que necesita, consiguiendo que sea independiente de la máquina en la que se ejecuta.

## Kubernetes

Kubernetes es un sistema de código abierto destinado al manejo y orquestación de aplicaciones software en contenedores. Trabaja junto a los entornos de ejecución de contenedores, como Docker, para gestionar y automatizar el despliegue del software. [4]

Como hemos visto durante este capítulo, muchas de las herramientas MLOps trabajan junto a Kubernetes para realizar el despliegue de los modelos. Aunque estos modelos pueden simplemente desplegarse en un contenedor de Docker, en el momento en que manejamos varios modelos en distintos contenedores, conviene introducir Kubernetes, ya que nos permite crear un clúster en el que desplegar y mantener todos nuestros modelos de forma conjunta.

El uso de Kubernetes para el despliegue de modelos tiene varias ventajas. Una de ellas es la alta disponibilidad, ya que nos permite crear copias de nuestra aplicación (en nuestro caso los modelos) para que, en caso de que una de las copias deje de funcionar, se redirija el tráfico a las demás. Este factor también proporciona una escalabilidad que utilizando únicamente Docker no podríamos conseguir. Otro beneficio que aporta es la capacidad de recuperación ante errores.

## GitHub Actions

GitHub Actions es una herramienta destinada a la integración y al despliegue continuo de software que, como su nombre indica, forma parte del famoso repositorio online GitHub. Permite crear y ejecutar flujos de trabajo que se ejecutan cuando sucede cierto evento.

Esta herramienta funciona de forma que, cuando se detecta algún evento nuestro

repositorio en GitHub, se ejecuta el flujo de trabajo que previamente hemos determinado para ese evento. Este flujo estará formado por una serie de acciones, que pueden ser creadas por nosotros mismos o pueden ser reutilizadas de otros usuarios que las crean y publican en GitHub Marketplace.

Los eventos que desencadenan la ejecución del flujo de trabajo pueden ser casi cualquier cosa que ocurra en el repositorio: que se haga un push sobre una de sus ramas, que se elimine algún fichero, que se envíe una petición de cambio, que se modifique una etiqueta... Entre muchos otros. [5]

Como hemos visto, MLOps se basa en la automatización del flujo de trabajo de los modelos de machine learning, por lo que esta herramienta puede sernos de gran utilidad a la hora de aplicar la integración continua y la entrega continua durante el desarrollo y la puesta en producción de modelos de machine learning.

## 2.4. Conclusiones

Como hemos visto, existe una alta variedad de herramientas que aplican el enfoque MLOps para conseguir que las distintas fases del ciclo de vida de proyectos ML sean más eficientes y eficaces. La elección de herramientas dependerá directamente de las fases del ciclo que queramos automatizar, así como del tipo de entorno en el que queramos aplicar MLOps.

Algunas herramientas están más enfocadas a científicos de datos, que no tienen por qué conocer los procesos propios de la Ingeniería del Software. Otras herramientas están más enfocadas a equipos centrados en la integración continua y la entrega continua.

En definitiva, las claves para que la aplicación de MLOps en un proyecto sea satisfactoria son: conocer las características del entorno en el que se quiere aplicar y determina las fases del ciclo de vida que se quieren abarcar. En nuestro caso, partimos de un escenario en el que nos interesa automatizar el despliegue de modelos de machine learning, por lo que tendremos que centrarnos en las herramientas que se centran en esta fase.

---

## 3. Objetivos y metodología

---

### 3.1. Introducción

En este capítulo se explicará más en profundidad cómo se ha enfocado el desarrollo de este proyecto. Definiremos los objetivos a cumplir en cada una de las fases del proyecto: análisis, diseño y desarrollo. También comentaremos las metodologías seguidas en cada una de las fases.

### 3.2. Definición de objetivos

Tras familiarizarnos con MLOps y con las herramientas que lo implementan, tenemos una idea general de cómo podemos abordar un proyecto de machine learning aplicando esta disciplina. Llega entonces el momento de definir los objetivos de este trabajo más allá de conocer las técnicas de Machine Learning Operations.

En definitiva, este trabajo parte de un escenario concreto sobre el que queremos aplicar MLOps. Una vez que tenemos la visión general del estado de la tecnologías, podemos centrarnos en este escenario, tomándolo como un caso de estudio a analizar. Esto se abordará en la fase de análisis del proyecto.

Tras el análisis, tendremos más claro lo que queremos conseguir con nuestro sistema MLOps y las herramientas que nos pueden ayudar a conseguirlo. Toca pasar a la fase de diseño, en la que idearemos la solución MLOps basándonos en la funcionalidad básica definida en la fase de análisis. El objetivo final es diseñar una infraestructura que cumpla con las necesidades de los usuarios presentes en el escenario.

Por último, una vez se haya diseñado una solución, pasaremos a la fase de desarrollo, en la que tendremos que probar que la infraestructura puede desplegarse y utilizarse para cumplir con la funcionalidad básica propuesta.

En definitiva, el objetivo global del proyecto es familiarizarnos con las tecnologías de MLOps mediante la construcción de una infraestructura que de soporte a la asignatura de Aprendizaje Automático. La parte de MLOps que nos interesa no tiene que ver con el desarrollo del modelo, ya que de eso se encargarán los alumnos de manera desacoplada al resto. El sistema abarcará el despliegue y monitorización de modelos únicamente. Para cumplir este objetivo global, vamos a definir varios objetivos particulares:

- Familiarizarnos con las herramientas disponibles.
- Definir un escenario concreto para un prototipo.



- Probar las herramientas que pueden encajar en este escenario.
- Desarrollar scripts básicos que nos ayuden a automatizar procesos.

### 3.3. Enfoque del análisis

En esta fase de análisis se estudiarán las características del escenario propuesto, se definirán distintos roles entre los usuarios y se determinarán los principales casos de uso. Dentro del análisis de las características del escenario, la parte más importante será determinar qué fases del flujo de trabajo ML queremos automatizar. También estimaremos el tamaño del sistema que queremos diseñar teniendo en cuenta el número de alumnos y de modelos por alumno.

En el caso de nuestro escenario, queremos automatizar el despliegue de los modelos, por lo que tendremos que centrarnos en las herramientas que abarcan esta fase del ciclo de vida.

En conclusión, el objetivo es conocer las características del caso de estudio ante el que nos encontramos y determinar la funcionalidad que debe aportar el sistema MLOps a desarrollar. El método utilizado para determinar la funcionalidad del sistema se ha basado en la detección de distintos roles entre los usuarios del sistema y en la determinación de los casos de uso de cada uno de estos roles.

### 3.4. Enfoque del diseño

En esta fase del diseño nos centraremos en la elección de las herramientas que nos van a permitir crear un sistema que cumpla con la funcionalidad propuesta en la fase de análisis. Analizaremos el funcionamiento de estas herramientas y planificaremos la integración de todas ellas para obtener una solución MLOps adecuada al caso de estudio.

El proceso de diseño se ha llevado a cabo teniendo en cuenta la funcionalidad determinada en el capítulo de análisis, especialmente aquella funcionalidad asociada a los casos de uso que más se han desarrollado.

### 3.5. Enfoque del desarrollo

En esta fase nos centraremos en la puesta en marcha del sistema MLOps en un entorno de prueba para comprobar su funcionamiento a la hora de desplegar modelos de aprendizaje automático reales. Esta primera versión del sistema deberá cumplir con los casos de uso esenciales propuestos en la fase de análisis.

---

## 4. Análisis

---

### 4.1. Introducción

En este capítulo analizaremos más en profundidad el escenario del que parte este proyecto. Determinaremos las características del escenario y las necesidades de los usuarios que en él se presentan. Concretaremos las fases del ciclo de vida en la que nos interesa aplicar el enfoque MLOps para asentar un precedente que nos servirá más adelante para realizar la selección de herramientas.

Para estudiar el escenario se seguirá un enfoque propio de la Ingeniería del Software, basado en la definición de distintos actores que interactúan con el según una serie de casos de uso. Un caso de uso es la descripción de una acción que realiza un usuario para llevar a cabo un proceso en el sistema.

### 4.2. Descripción del escenario

El escenario a analizar es el siguiente:

Un alumno de la asignatura Aprendizaje Automático de cuarto curso de Ingeniería Informática termina de crear un modelo de machine learning y quiere poder desplegarlo para que otros usuarios puedan usarlo, especialmente su profesor, que tendrá que evaluar su trabajo. Coge el software desarrollado y lo sube a un repositorio. Automáticamente, este modelo se empaqueta y se publica en una web desde la que los usuarios pueden hacer peticiones y recibir respuestas del modelo.

Los usuarios del sistema podrán acceder a los modelos sin necesidad de desplegarlos localmente uno a uno. En el caso concreto de este escenario, el profesor que quiere corregir el trabajo realizado por sus alumnos podrá acceder como usuario a la web y enviar peticiones a los distintos modelos para evaluar su funcionamiento. Todos los modelos se gestionarán en un mismo lugar y se ejecutarán en igualdad de condiciones.

Para conseguir lo que se propone en este escenario tendremos que aplicar el enfoque MLOps, logrando que los alumnos puedan desplegar y probar los modelos sin necesidad de conocer la infraestructura o programas involucrados en el proceso. Ellos solo tendrán que preocuparse de aportar el código fuente con la estructura y formato adecuados, el resto se hará automáticamente.

En conclusión, como podemos ver, la fase del ciclo de vida del proyecto que tenemos que automatizar es el despliegue de los modelos, por lo que nos tendremos que centrar en las herramientas que ofrezcan esta posibilidad.

A la hora de realizar el diseño también tendremos que tener en cuenta el tamaño que deberá tener el sistema para alojar todos los modelos de los alumnos. Estimaremos unos 60 alumnos por año, que a su vez generarán no más de tres modelos. Teniendo estos factores en cuenta, nuestro sistema deberá poder alojar unos 180 modelos simultáneamente.

### 4.3. Usuarios del sistema

En este escenario se pueden distinguir tres usuarios base del sistema: el administrador, encargado de configurar y mantener la infraestructura; los alumnos, encargados de producir los modelos de machine learning y de subirlos al repositorio; los consumidores de modelos, que podrán realizar peticiones y obtener respuestas de los mismos.

Administrador	Gestiona y mantiene la infraestructura detrás del sistema MLOps
Alumno	Produce modelos y los publica en un repositorio para que se desplieguen automáticamente.
Consumidor de modelos	Envía peticiones a los modelos.

Cuadro 4.1: Usuarios del sistema

En nuestro caso, el profesor de la asignatura será el principal consumidor de los servicios de los modelos, ya que accederá al sistema para evaluar el trabajo de sus alumnos. Sin embargo, otro ejemplo de consumidor podría ser una aplicación externa que haga uso de los modelos para explotarlos.

### 4.4. Casos de Uso

En este apartado presentaremos los principales casos de uso a tener en cuenta a la hora de diseñar nuestro sistema MLOps. Un caso de uso es una breve descripción de una actividad que realiza un usuario sobre el sistema. Estas descripciones nos servirán para asentar las bases sobre las que diseñaremos nuestro sistema.

#### 4.4.1. Administrador

Estos son los principales casos de uso del administrador en el sistema MLOps:

1. El administrador da de alta alumnos.
2. El administrador gestiona activa el repositorio de modelos.
3. El administrador desactiva el repositorio de modelos.

4. El administrador cancela el despliegue de un modelo.
5. El administrador consulta los modelos desplegados.
6. El administrador gestiona su panel de control.

De estos casos de uso, prestaremos especial atención al caso de uso número 6. Tendremos que asegurarnos de que el administrador del sistema puede acceder a un panel de control desde el que consultar los modelos desplegados y su estado.

#### **4.4.2. Alumno**

Estos son los principales casos de uso del alumno en el sistema MLOps:

1. El alumno genera un modelo y lo sube al sistema para que se despliegue.
2. El alumno consulta los modelos que tiene subidos en el sistema.
3. El alumno consulta los usos que se han hecho de sus modelos.

En esta caso, prestaremos especial atención al caso de uso número 1, ya que es la funcionalidad principal que deberá ofrecer el sistema MLOps a desarrollar. Los alumnos deben poder subir sus modelos a un repositorio para que se desplieguen automáticamente.

#### **4.4.3. Consumidor de modelos**

Estos son los principales casos de uso del consumidor de modelos en el sistema MLOps:

1. El consumidor de modelos consulta los modelos disponibles.
2. El consumidor de modelos envía una petición al modelo.

El caso de uso más importante en esta ocasión es el número 2. Al igual que en el caso anterior, este caso de uso pertenece a la funcionalidad básica del sistema que queremos desarrollar. Los sistemas o usuarios externos deben poder realizar consultas sobre los modelos a través de sus API.

---

## 5. Diseño y desarrollo

---

### 5.1. Introducción

En este capítulo se expondrá el diseño e implementación de la solución MLOps destinada a facilitar el despliegue de modelos de machine learning en el contexto del escenario descrito en el capítulo 4 de Análisis. Presentaremos la infraestructura desarrollada para desplegar, gestionar y mantener los modelos de aprendizaje automático, así como el proceso de automatización del flujo de trabajo. Al tratarse de un sistema formado por varias herramientas, profundizaremos en el funcionamiento individual de cada una de ellas, así como en la integración conjunta de las mismas.

Por último, presentaremos las pruebas de despliegue realizadas con un modelo de machine learning real. Se trata de un modelo para predecir la diabetes creado en lenguaje R. Utiliza un *dataset* de la librería *mlbench* que contiene los resultados de análisis médicos realizados sobre sujetos de la tribu indígena Pima.

### 5.2. Selección de herramientas

Las herramientas que finalmente han sido seleccionadas para diseñar el sistema MLOps en este escenario son las siguientes: Seldon Core, Docker, Kubernetes y GitHub Actions. A continuación vamos a explicar más en profundidad el funcionamiento de estas herramientas a la hora de desplegar los modelos de machine learning.

#### 5.2.1. Seldon Core

Como vimos en el Capítulo 2 sobre el estado del arte, Seldon Core es una plataforma que nos va a permitir poner en producción nuestros modelos de aprendizaje automático como microservicios de tipo REST/GRPC.

En el caso de nuestro escenario, queremos conseguir que, al desplegar un modelo previamente entrenado, se puedan enviar peticiones a su función `predict` tal y como se hace con cualquier API REST de la web. El modelo generará una respuesta que enviará en forma de documento JSON.

Finalmente, hemos seleccionado esta herramienta frente a las otras opciones destinadas al despliegue y monitorización de modelos porque nos ofrece una solución directa para desplegar modelos en Python y en R. Las otras opciones parecen estar más ligadas a trabajar en conjunto con frameworks concretos de desarrollo de modelos.

Seldon Core ofrece soluciones para el despliegue de distintos tipos de proyectos ML. Por un lado, ofrece varios servidores de modelos preempaquetados que pueden usarse para desplegar modelos previamente entrenados y serializados usando ciertas bibliotecas de aprendizaje automático, como pueden ser Scikit-learn o TensorFlow.

Sin embargo, nuestro objetivo es, principalmente, desplegar modelos de machine learning basados en métodos predictivos en R y que utilizan librerías propias de este lenguaje. La mejor opción que nos ofrece Seldon Core para estos casos son los "wrappers" de lenguaje.

## Language Wrappers

Los "wrappers" de lenguaje son, en definitiva, una pequeña capa de código que tiene como función "traducir" la interfaz de un software ya existente (en un lenguaje específico) para aumentar su compatibilidad con otros elementos software. En este caso, se pretende envolver un modelo en lenguaje R para que pueda ser compatible con el proceso de despliegue de Seldon Core.

Para envolver nuestros modelos en R, Seldon Core propone utilizar la herramienta Source-to-Image (S2I) de OpenShift. Esta herramienta permite crear imágenes que pueden ser desplegadas en contenedores (como Docker) a partir de código fuente. A partir de un imagen constructora, que en nuestro caso será la imagen para modelos R de Seldon Core, y un directorio que contenga el modelo que queremos empaquetar (ver estructura del directorio 5.6), podremos construir la imagen de nuestro modelo lista para ser desplegada. Esto lo conseguiremos ejecutando el siguiente comando:

```
$ s2i build . seldonio/seldon-core-s2i-r:0.1 mymodel
```

En este comando **seldonio/seldon-core-s2i-r:0.1** es la imagen constructora para modelos R de Seldon Core, mientras que **mymodel** es el nombre de la imagen que vamos a construir con nuestro modelos.

## Técnicas de despliegue

Una vez que hemos creado la imagen de nuestro modelo, nos encontramos con dos formas de desplegarlo: una más simple, que utiliza únicamente Docker; otra más robusta y escalable, que utiliza también Kubernetes.

### Contenedores de Docker

La primera opción, que utiliza únicamente Docker, consiste en crear un contenedor local de Docker que contenga la imagen de nuestro modelo, de forma que puede ejecutarse y quedar en escucha de posibles peticiones. Para esta opción, simplemente tenemos que tener el cliente de Docker instalado y ejecutar un comando para correr la imagen del modelo en un contenedor. Supongamos que tenemos una imagen de modelo

llamada *mymodel*. La forma de correr este modelo sería utilizar el comando **docker run -p 5000:5000 -rm mymodel**. El API del modelo quedará disponible y a espera de peticiones a través de **localhost:5000/predict**.

Esta opción, al ser tan simple, no proporciona la capacidad de gestión, robustez y posibilidad de escala que tiene la opción de despliegue mediante un clúster de Kubernetes. En caso de que nuestra aplicación reciba muchas peticiones, no conviene que solo exista una instancia desplegada de nuestro modelo, por lo que esta opción no sería la más adecuada. Sin embargo, si queremos simplemente probar un modelo o comprobar que la imagen se ha generado correctamente, este método es una buena forma de testear el punto de acceso a un modelo.

### Clúster de Kubernetes

Como hemos visto anteriormente, Seldon Core se presenta como una herramienta para desplegar modelos utilizando Kubernetes. Esta es, definitivamente, la forma más completa de realizar el despliegue. La idea es crear un clúster de Kubernetes, ya sea de forma local o en la nube, en el que realizaremos una serie de instalaciones y configuraciones para que nuestros modelos puedan desplegarse y mantenerse de una forma más efectiva.

Además, Seldon Core proporciona una interfaz de usuario estandarizada, creada mediante una especificación OpenAPI/Swagger, para enviar peticiones a cada modelo que despleguemos en nuestro clúster de Kubernetes.

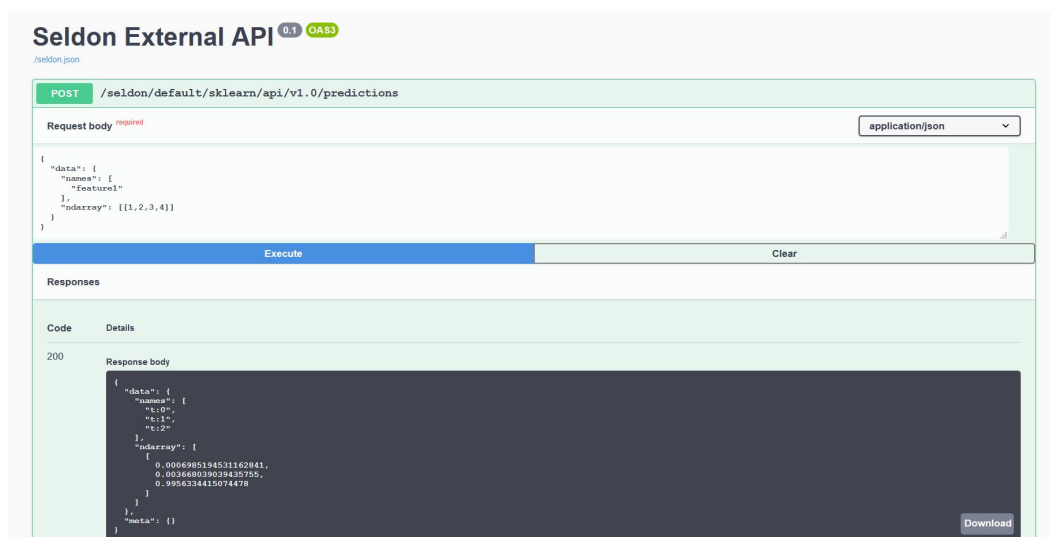


Figura 5.1: Interfaz estandarizada para enviar peticiones a modelos. Fuente: [docs.seldon.io](https://docs.seldon.io)

### 5.2.2. Kubernetes

Kubernetes va a ser una parte muy importante de nuestra solución MLOps. En esta sección se presentarán algunos conceptos clave del ámbito de trabajo de Kubernetes que conviene que conozcamos para entender el funcionamiento del clúster que vamos a desplegar.

Por un lado, tenemos los objetos propios de Kubernetes, entre los que se encuentran los *namespaces* o espacios de nombre y los *pods*. Los espacios de nombres son, en definitiva, clústeres virtuales dentro del clúster físico. Sirven para marcar un campo de acción para los nombres de los recursos (pueden haber recursos con el mismo nombre siempre que estén en espacios distintos) y para dividir los recursos del clúster entre múltiples usuarios. Los pods son grupos de uno o más contenedores con almacenamiento o red compartidos. En nuestro caso, Seldon Core se encargará de que cada imagen de modelo sea desplegada en un pod independiente.

Por otro lado, tenemos el concepto de nodo, que es la máquina virtual encargada de ejecutar las aplicaciones contenidas en los contenedores. Por simplicidad, nosotros trabajaremos con un solo nodo, pero en caso de querer crear un sistema de despliegue de modelos a gran escala, sería conveniente contar con distintos nodos que contengan distintas instancias de los pods para que, en caso de que falle uno de los nodos, se pueda recurrir al otro.

En la Figura 5.2 podemos ver una representación gráfica de algunos de los elementos de Kubernetes que hemos presentado.

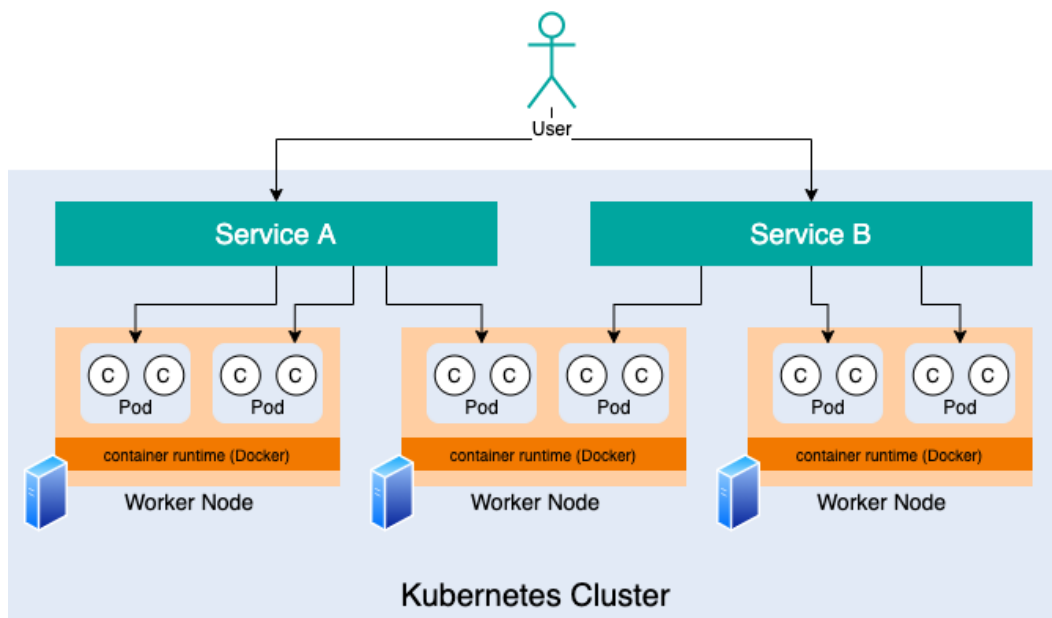


Figura 5.2: Clúster de Kubernetes. Fuente: [www.theserverside.com](http://www.theserverside.com)

Otro punto clave a destacar es el concepto de operador. En definitiva, un operador de Kubernetes es un elemento software que contiene el conocimiento de cómo actuar



ante una situación. En nuestro caso, tendremos que instalar el operador de Seldon Core, que se encargará de facilitar la producción, monitorización y mantenimiento de los sistemas de machine learning a escala.

Además de este operador, también instalaremos en nuestro clúster un *service mesh* o malla de servicios. Una malla de servicio es una pieza software que se encarga de controlar el intercambio de datos entre las distintas partes de una aplicación. Son especialmente necesarias en los sistemas que siguen una arquitectura basada en microservicios. Estos servicios independientes deben seguir unas normas de comunicación, que deben ser configuradas en cada uno de ellos. La malla de servicio es, en definitiva, una capa de la infraestructura que contiene las normas que rigen la comunicación entre los servicios, de forma que no hay que configurar cada servicio por separado. [9]

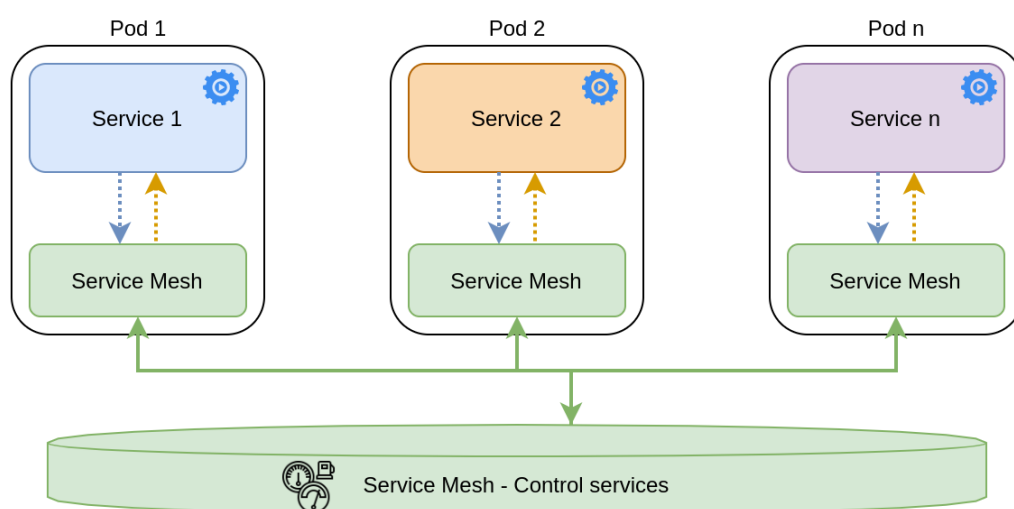


Figura 5.3: Representación gráfica de una malla de servicio. Fuente: [www.teldat.com](http://www.teldat.com)

En el caso de nuestro escenario, cada modelo se desplegará en el clúster como un microservicio. En un inicio, si se gestiona un número limitado de modelos, no tendría por qué ser necesario utilizar una malla de servicios. Sin embargo, conviene añadir esta capa en nuestro sistema para garantizar la escalabilidad en caso de aumentar el número de modelos alojados en el clúster.

En Kubernetes, los despliegues se realizan mediante un fichero JSON o YAML. En este fichero tendremos que indicar los recursos que tendrá el contenedor en el que desplegaremos nuestra aplicación. Entre estos recursos pueden aparecer características como la capacidad de procesamiento o la memoria. El operador de Seldon Core amplía este aspecto de Kubernetes con un tipo customizado de recurso, llamado *SeldonDeployment*, que nos va a permitir definir grafos de inferencia con nuestros modelos.

La principal tarea del operador de Seldon Core será leer la definición de los recursos de tipo *SeldonDeployment* para asegurarse de que se crean todos los componentes necesarios para el despliegue, como los pods.

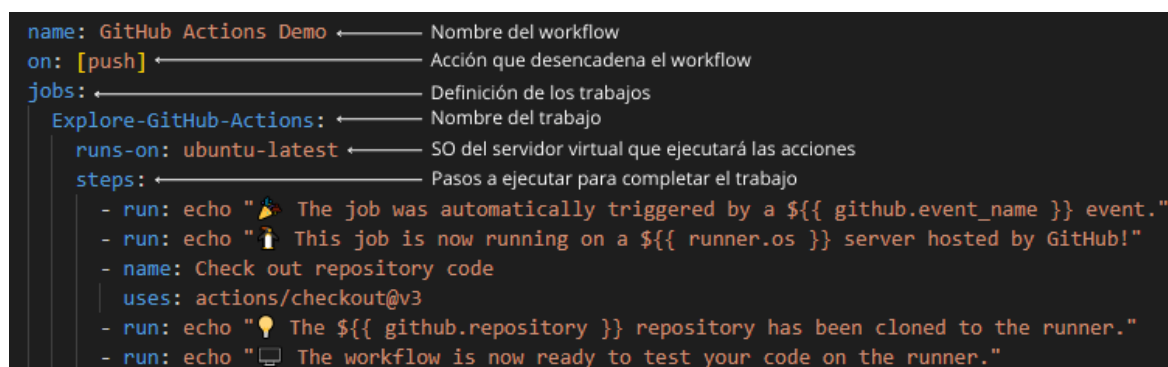
### 5.2.3. GitHub Actions

Hasta ahora, todo lo que hemos visto es el proceso manual para crear la imagen de los modelos y desplegarlos en el clúster de Kubernetes. Todo esto, como veremos en la sección 5.4, se consigue mediante comandos propios de las distintas herramientas: `s2i` para la creación de imágenes, `kubectl` para la gestión del clúster de Kubernetes, `docker` para la gestión de las imágenes... Llegados a este punto nos encontramos con un nuevo reto: automatizar el proceso para que, cada vez que se genera un modelo, este se empaquete y se despliegue en nuestro clúster automáticamente. Para esta tarea se ha seleccionado la herramienta GitHub Actions.

Como vimos en el capítulo 2, GitHub Actions es una herramienta destinada a la integración continua y la entrega continua de software, que en nuestro caso serán modelos de machine learning. Una de las grandes ventajas que posee esta herramienta es que está integrada en el famoso repositorio web GitHub. La gran mayoría de alumnos, aunque probablemente no estén familiarizados con GitHub Actions directamente, sí que lo estarán con el funcionamiento de los repositorios de GitHub.

La idea es conseguir que los alumnos solo tengan que conocer como utilizar un repositorio. Cuando actualicen el repositorio con un modelo, se desencadenarán una serie de acciones que completarán todo el proceso de creación de imágenes y despliegue de modelos. Queremos conseguir que los alumnos puedan abstraerse de la complejidad de la infraestructura que hay detrás. No tendrán que estar familiarizados con los elementos propios de Docker, Kubernetes o Seldon Core, simplemente tendrán que limitarse a crear sus modelos en un formato específico y subirlos a la web.

Para definir las acciones con GitHub Actions tan solo tenemos que crear un directorio llamado **.github/workflows** en la raíz de nuestro repositorio. Dentro de este directorio definiremos los workflow que se van a desencadenar cuando se produzca un determinado cambio en el repositorio. Estos workflows se definen mediante un fichero YAML en el que se indican las acciones a ejecutar. En la Figura 5.4 aparece un ejemplo de fichero YALM con el formato necesario para ser procesado por GitHub Actions. En la sección 5.4.2 veremos más en profundidad el funcionamiento de esta herramienta.



```
name: GitHub Actions Demo
on: [push]
jobs:
  Explore-GitHub-Actions:
    runs-on: ubuntu-latest
    steps:
      - run: echo "🎉 The job was automatically triggered by a ${{ github.event_name }} event."
      - run: echo "📄 This job is now running on a ${{ runner.os }} server hosted by GitHub!"
      - name: Check out repository code
        uses: actions/checkout@v3
      - run: echo "💡 The ${{ github.repository }} repository has been cloned to the runner."
      - run: echo "🚀 The workflow is now ready to test your code on the runner."
```

The image shows a code editor with a GitHub Actions workflow file. The code is in YAML format. To the right of the code, there are arrows pointing to specific lines with Spanish annotations explaining the purpose of each field:   
- `name: GitHub Actions Demo` is labeled "Nombre del workflow".   
- `on: [push]` is labeled "Acción que desencadena el workflow".   
- `jobs:` is labeled "Definición de los trabajos".   
- `Explore-GitHub-Actions:` is labeled "Nombre del trabajo".   
- `runs-on: ubuntu-latest` is labeled "SO del servidor virtual que ejecutará las acciones".   
- `steps:` is labeled "Pasos a ejecutar para completar el trabajo".   
The workflow steps include:   
1. A run action that prints a message about the event.   
2. A run action that prints the runner's OS.   
3. A named step "Check out repository code" using the `actions/checkout@v3` action.   
4. A run action that prints the repository name.   
5. A run action that prints a message about the workflow being ready.

Figura 5.4: GitHub Actions - Ejemplo de fichero YALM

### 5.3. Funcionamiento del sistema

En esta sección presentaremos la infraestructura completa del sistema, creado gracias a la integración de las herramientas comentadas en la sección anterior. En la Figura 5.5 podemos ver una representación de la infraestructura diseñada. El flujo de trabajo es el siguiente:

1. El repositorio de GitHub se actualiza con un nuevo modelo de machine learning. Al detectarse el evento "push" sobre el repositorio, se activa el flujo de trabajo de GitHub Actions.
2. GitHub Actions se encarga crear un servidor virtual en el que se ejecutarán los trabajos previamente definidos por el administrador. Primero se creará la imagen del modelo utilizando la herramienta S2I a partir del código subido al repositorio. Después de esto, se subirá esta imagen al repositorio web de imágenes DockerHub.
3. A continuación, se conectará con el clúster de Kubernetes. Este clúster deberá estar, o bien en un servidor o máquina externa, o bien en la nube.
4. Desde la máquina que contiene el clúster se realizará el "pull" de la imagen alojada en DockerHub. Por último, se realizará el despliegue del modelo en un pod del clúster.

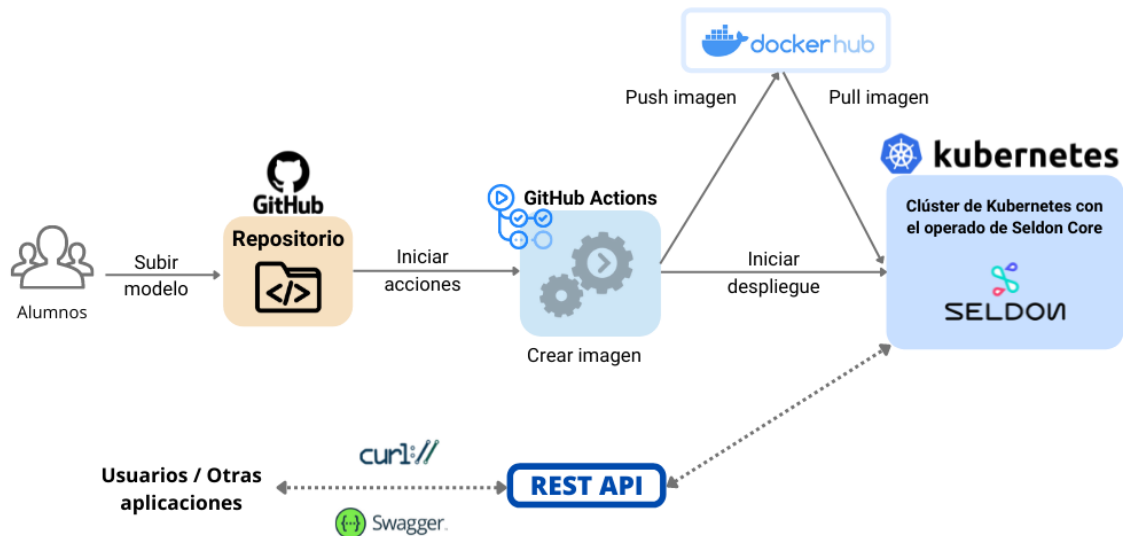


Figura 5.5: Infraestructura del sistema MLOps

Para poder entender mejor el funcionamiento del sistema, dividiremos el proceso en tres fases que serán explicadas con mayor detenimiento: subida de un modelo al repositorio, ejecución de acciones (creación de imagen) y despliegue del modelo en Kubernetes.

### 5.3.1. Fase I. Un modelo se sube al repositorio.

Los alumnos, tras haber creado y entrenado sus modelos de machine learning, tendrán que crear un directorio con el código necesario para ejecutar el modelo. Este directorio deberá seguir cierto patrón para que la herramienta S2I pueda crear la imagen del modelo. Dependiendo del lenguaje utilizado, los requisitos de este repositorio serán ligeramente distintos pero, por norma general, habrá que incluir una clase que muestre la lógica de nuestro modelo, un archivo con las dependencias y un directorio llamado **.s2i** que contenga un fichero **environment** en el que se indicarán algunos parámetros necesarios para la creación de la imagen. En la siguiente sección (5.4) veremos más en detalle como prepara el código para la creación de la imagen del modelo.

Cuando el alumno haga el push del directorio que ha creado con todos los elementos necesarios, se activará el workflow que previamente habrá sido definido en un fichero YAML en la ruta **.github/workflows**.

### 5.3.2. Fase II. Ejecución de trabajos con GitHub Actions.

GitHub Actions se encargará crear un servidor virtual que ejecutará los distintos trabajos del workflow. Este flujo de trabajo incluirá los siguientes pasos:

- Instalar la herramienta S2I utilizando una acción predefinida de GitHub Actions.
- Ejecutar el comando de creación de imágenes con S2I, obteniendo todos los recursos necesarios del propio repositorio.
- Utilizar una acción predefinida de Docker para subir una imagen a DockerHub (habrá que configurar el usuario y la contraseña)
- Conectar con el servidor (cloud u on premise) que aloja el clúster de Kubernetes para que inicie el despliegue.

### 5.3.3. Fase III. Despliegue en Kubernetes.

El proceso pasará entonces a realizarse en el servidor que aloja el clúster de Kubernetes con el operador de Seldon Core.

El primer paso será descargar la imagen del modelo desde el repositorio de DockerHub. Una vez se tenga la imagen en local, se utilizará un fichero YAML con la definición del SeldonDeployment en el que aparecerán las características del pod y los contenedores a crear para desplegar el modelo, utilizando el siguiente comando:

```
$ kubectl apply -f fichero-seldon-deployment.yml
```

Después de esto, el modelo estará accesible a través de un endpoint, a espera de recibir peticiones. Todo este proceso se entenderá mejor cuando en la siguiente sección (5.4) expliquemos el proceso de despliegue de un modelo real.

## 5.4. Pruebas realizadas

En este apartado explicaremos los pasos que se han seguido para poner en marcha el sistema, así como las pruebas de despliegue que se han llevado a cabo utilizando modelos de aprendizaje automático reales. En este caso, nos hemos centrado en el despliegue de un modelo desarrollados en R.

Concretamente, el modelo utilizado para las pruebas está destinado a la predicción de la diabetes. Este modelo ha sido entrenado con un *dataset* de la librería **mlbench** que contiene los resultados de análisis médicos realizados sobre sujetos de la tribu indígena Pima. Una vez entrenado, el modelo es capaz de predecir si un sujeto es positivo o negativo en diabetes a partir de los valores de sus pruebas médicas.

Cabe destacar que, como las pruebas para la puesta en marcha del sistema se han realizado en un ordenador personal con IP dinámica, se ha utilizado una herramienta extra, llamada Ngrok, para poder acceder en remoto al clúster de Kubernetes desde la máquina virtual de GitHub Actions. Esta herramienta nos permite asociar un servidor local a un subdominio, de forma que se puede acceder a él desde fuera de su propia LAN. Esta solución fue pensada únicamente para poder realizar pruebas de funcionamiento básicas. En una puesta en marcha real del sistema, habría que valorar dónde queremos alojar el clúster de Kubernetes que contendrá todos los modelos de los alumnos.

Para poder plasmar más fácilmente todos los pasos realizados para la puesta en marcha del sistema, dividiremos esta sección en varios apartados: preparación de la máquina que aloja el clúster de Kubernetes (5.4.1), preparación del modelo para el despliegue (5.4.2) y declaración del flujo de trabajo en GitHub Actions (5.4.3).

### 5.4.1. Preparación del clúster

Como cabe esperar, el primer paso de las pruebas fue crear un clúster de Kubernetes alojado en local, instalando en él el operador de Seldon Core. La instalación y configuración de las distintas herramientas necesarias realizar el despliegue se explicará en el Anexo A.

El siguiente paso fue crear un *script* con los pasos que deberán ejecutarse en la máquina una vez que la imagen del modelo se haya subido a DockerHub. Estos pasos consisten simplemente en descargar la imagen de DockerHub y crear el despliegue mediante un fichero YAML. Podemos observar el fichero de despliegue y el *script* en los extractos de código 5.2 y 5.1.

```

1  docker pull mjbernal/mymodel:latest
2  kubectl create namespace mynamespace
3  kubectl apply -f ./deploy.yml

```

Extracto de código 5.1: Fichero **simple.script**

```

1  apiVersion: machinelearning.seldon.io/v1
2  kind: SeldonDeployment
3  metadata:
4    name: mymodel
5    namespace: mynamespace
6  spec:
7    name: mymodel
8    predictors:
9    - componentSpecs:
10      - spec:
11        containers:
12        - name: classifier
13          image: mjbernal/mymodel:latest
14      graph:
15        name: classifier
16      name: default
17      replicas: 1

```

Extracto de código 5.2: Fichero **deploy.yml**

Como hemos dicho, para que el servidor virtual que crea GitHub Actions pueda acceder a la máquina local utilizamos la herramienta Ngrok. Ejecutando el comando **ngrok tcp 22**, obtenemos un subdominio a través del cuál podremos conectarnos a nuestra máquina desde el exterior de la red utilizando SSH. Debemos guardar este subdominio, ya que más tarde tendremos que utilizarlo para configurar el flujo de trabajo de GitHub Actions.

Una vez que tenemos el clúster configurado, el script y fichero de despliegue preparados y el puerto 22 TCP disponible, podemos pasar a los siguientes pasos: configurar el repositorio de GitHub, preparar el modelo y los recursos necesarios y crear el fichero con la definición de los trabajos a realizar cuando se haga un push sobre el repositorio.

### 5.4.2. Preparación del repositorio y del modelo

El siguiente paso fue crear el repositorio de GitHub en el que se subirían los recursos necesarios para el despliegue de modelos. En la raíz de este repositorio se incluyó el directorio **.github/workflows**, en el que debemos crear un fichero YAML con los trabajos a ejecutar cada vez que se haga un push en el repositorio.

A continuación se explicará el formato que tienen que tener el modelo y los recursos a la hora de introducirlos en el repositorio. S2I tiene que ser capaz de crear la imagen a partir de ellos, por lo que es importante el contenido que subamos al repositorio siga cierto patrón.

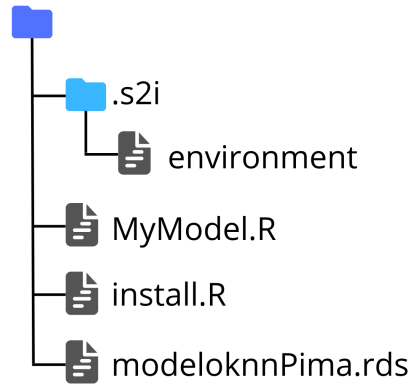


Figura 5.6: Formato del directorio

En el caso de los modelos en lenguaje R, tenemos que incluir los siguientes elementos (Ver Figura 5.6):

- El modelo ya entrenado y serializado en un fichero con formato .rds.
- Un fichero en R que proporcione una clase S3 que defina la lógica del modelo mediante una función predict. Ver fichero MyModel.R (5.3).
- Un fichero install.R con las librerías que necesita el modelo. Ver fichero install.R (5.4).
- Un directorio ./s2i que contenga un fichero environment (5.5) en el que se definan los parámetros necesarios para la creación de la imagen.

```

1  library(methods)
2  library(caret)
3  library(mlbench)
4
5  predict.mymodel <- function(mymodel, newdata=list()) {
6
7    # Extraer el individuo a consultar
8    individuo <- newdata[[1]][1]
9
10   # Cargar los datos y hacer la particion
11   data("PimaIndiansDiabetes")
12   pima.Datos = PimaIndiansDiabetes
13   pima.Var.Salida.Usada = c("diabetes")
14   set.seed(1234)
15   pima.TrainIdx.80 <- createDataPartition(pima.Datos[[pima.Var.Salida.
Usada]], p=0.8, list = FALSE, times = 1)
16   pima.Datos.Train <- pima.Datos[pima.TrainIdx.80,]
17
18   # Predict
19   predict(mymodel$model, pima.Datos.Train[individuo,])
20 }
21
22

```

```

23     new_mymodel <- function() {
24         model <- readRDS("modeloknnPima.rds")
25         structure(list(model=model), class = "mymodel")
26     }
27
28
29     initialise_seldon <- function(params) {
30         new_mymodel()
31     }

```

Extracto de código 5.3: Fichero **MyModel.R**

```

1     install.packages('rpart')
2     install.packages('mlbench')
3     install.packages('kknn')

```

Extracto de código 5.4: Fichero **install.R**

```

1     MODELNAME=MyModel.R
2     API_TYPE=REST
3     SERVICE_TYPE=MODEL
4     PERSISTENCE=0

```

Extracto de código 5.5: Fichero **environment**

A continuación vamos a explicar un poco más en profundidad el contenido del fichero **MyModel.R** (5.3). En él se define una función **initialise\_seldon** que se encarga de crear la clase S3 de nuestro modelo mediante el constructor **new\_model**. En este constructor se lee el modelo empaquetado en el fichero `.rds`. También se define una función genérica **predict** para la clase del modelo. Esta función será la encargada de recibir llamadas con datos para realizar una predicción. En nuestro caso, la función se encarga de cargar el conjunto de datos y buscar en ellos el individuo que se ha solicitado. Finalmente, se llama a la función **predict** del modelo con los datos del individuo.

El nombre de este fichero y el indicado en el parámetro **MODEL\_NAME** del fichero `./s2i/environment` (5.5) deberá ser el mismo.

### 5.4.3. Creación del flujo de trabajo con GitHub Actions

Una vez que tenemos todos estos ficheros, el siguiente paso es crear el fichero de definición del flujo de trabajo en el directorio `.github/workflows` del repositorio, para que cuando subamos nuestra carpeta con los recursos se desencadenen las acciones. En el extracto de código 5.6 podemos ver el fichero **cicd.yml**, en el que se definen los trabajos a ejecutar.



El primer paso del flujo de trabajo es instalar S2I en el servidor virtual y ejecutar el comando responsable de la creación de la imagen del modelo. Como podemos ver en la línea 24 del extracto de código 5.6, el nombre asignado a la imagen es *mjbernal/mymodel*, siendo *mjbernal* el usuario dueño del repositorio de DockerHub al que queremos subir la imagen y *mymodel* el nombre de la imagen en sí. Si queremos indicar la versión de la imagen, deberíamos añadir ":" seguido de la versión (Ejemplo: *mjbernal/mymodel:0.1*).

A continuación se inicia sesión en DockerHub utilizando una acción predefinida. Para poder realizar este paso, primero tendremos que configurar los credenciales de la cuenta de DockerHub en la sección de secretos cifrados de GitHub. Después de esto, se ejecuta un comando para subir la imagen al repositorio.

Por último, se utiliza el subdominio creado con Ngrok para acceder mediante SSH al clúster y lanzar el *script* que previamente habíamos creado. Para este paso también tendremos que configurar un secreto cifrado de GitHub con la contraseña de acceso a la máquina local.

```
1     name: ci
2
3     on:
4       push:
5         branches:
6           - "master"
7
8     jobs:
9       despliegue:
10        runs-on: ubuntu-latest
11        steps:
12
13          # Instalar s2i
14          - name: Instalar S2I
15            uses: redhat-actions/openshift-tools-installer@v1
16            with:
17              source: "github"
18              # Using GitHub token from the github context
19              github_pat: ${ github.token }
20              s2i: "1.2"
21
22          # Crear la imagen Docker del modelo con s2i
23          - name: Ejecutar s2i
24            run: s2i build https://github.com/mjbernal/prueba-gha.git --
context-dir=mymodel-R seldonio/seldon-core-s2i-r:0.1 mjbernal/mymodel
25
26          # Iniciar sesion en DockerHub
27          - name: Login to DockerHub
28            uses: docker/login-action@v1
29            with:
30              username: ${ secrets.DOCKER_USERNAME }
31              password: ${ secrets.DOCKER_PASSWORD }
32
33          # Subir la imagen del modelo a DockerHub
34          - name: Docker Push
```

```

35         run: docker push mjbernal/mymodel:latest
36
37         Conectarse a la maquina del host para descargar la imagen de
DockerHub
38         Crear/iniciar un contenedor Docker con ella
39         — name: Acceder mediante SSH al host o servidor donde queremos
desplegar el servicio REST
40         uses: garygrossgarten/github-action-ssh@release
41         with:
42             command: bash ./simple.script
43             host: 2.tcp.eu.ngrok.io # Direccion estatica temporal creada
con ngrok en el host
44             port: 15158
45             username: mj
46             password: ${ secrets.SSHPASSWORD }

```

Extracto de código 5.6: Fichero **cicd.yml**

Una vez que se ejecuta todo esto, el modelo deberá estar disponible a través de los puntos de acceso propios de los despliegues de Seldon Core. En el caso del modelo que hemos desplegado, los puntos de acceso son los siguientes:

- <http://127.0.0.1:8080/seldon/mynamespace/mymodel/api/v1.0/predictions>
- <http://127.0.0.1:8080/seldon/mynamespace/mymodel/api/v1.0/doc/>

El primero es a través del cuál podemos enviar peticiones directamente con Curl. El segundo es la dirección de la interfaz estandarizada Swagger, que permite enviar peticiones de una forma más amigable para el usuario.

#### 5.4.4. Peticiones a la API del modelo

A continuación se presentarán algunas peticiones realizadas al modelo desplegado. Por la forma en la que hemos definido la clase S3 para construir la imagen, tan solo tenemos que indicar la posición del individuo que queremos consultar en el *dataset*. En la Figuras 5.7 y 5.8 aparece un ejemplo de petición a través de la interfaz Swagger. Por otro lado, en la Figura 5.9 podemos ver una consulta realizada directamente con Curl.

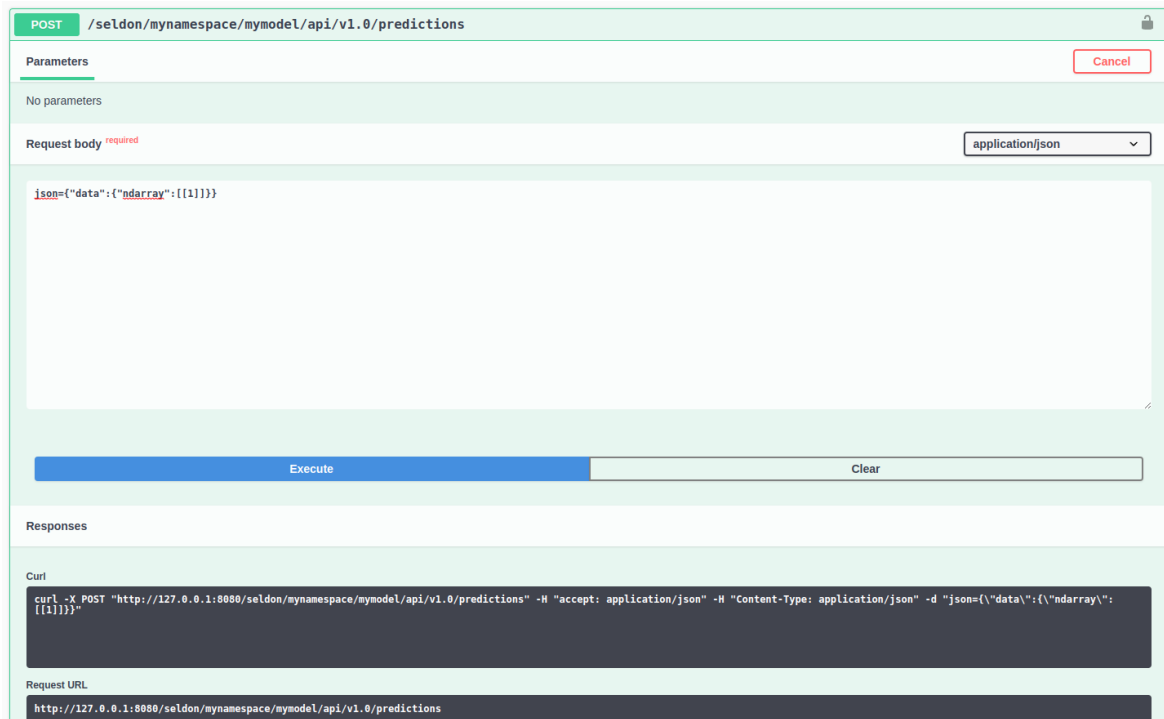


Figura 5.7: Petición del diagnóstico del individuo 1 mediante la interfaz Swagger

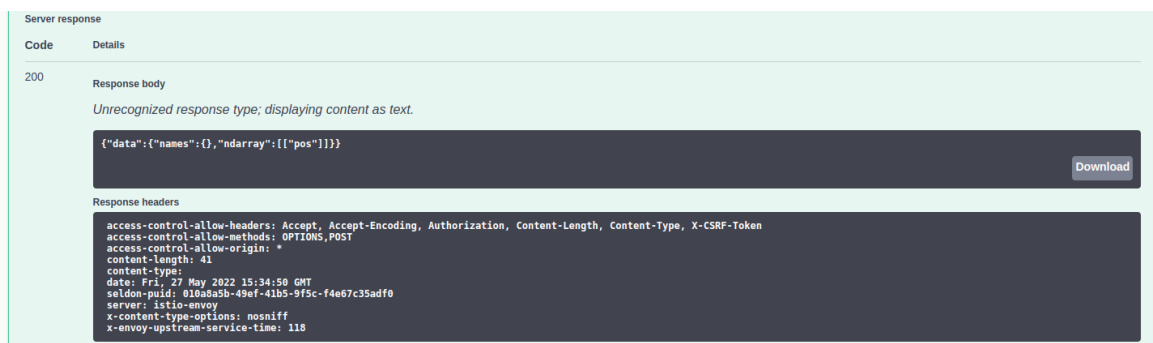


Figura 5.8: Respuesta a la petición de diagnóstico del individuo 1 a través de la interfaz Swagger

```
nj@mj-HP:~$ curl -X POST http://127.0.0.1:8080/seldon/mynamespace/mymodel/api/v1.0/predictions -H 'Content-Type: application/json' -d 'json={"data":{"ndarray":[[2]]}}'
{"data":{"names":{},"ndarray":[[\"neg\"]]}}
```

Figura 5.9: Petición de diagnóstico del individuo 2 y respuesta a través de Curl

---

## 6. Conclusiones y vías futuras

---

Este Trabajo de Fin de Grado supone un comienzo en la exploración de las soluciones MLOps para la gestión de los modelos de machine learning en el ámbito del departamento de Ingeniería de la Información y las Comunicaciones de la Facultad de Informática. En él hemos presentado los conceptos clave de Machine Learning Operations y de sus tecnologías asociadas. Al tratarse de una disciplina bastante nueva, día a día aparecen nuevas herramientas que aplican estas prácticas para abordar los proyectos de machine learning. Debido a esto, son muchas las opciones a explorar cuando queremos automatizar los procesos de nuestros proyectos. En este trabajo se ha querido plasmar el estado actual de las tecnologías para que, a la hora de querer aplicar MLOps sobre un proyecto de machine learning en particular, se cuente con una base sobre la que empezar a diseñar una solución.

En este trabajo se ha diseñado una solución concreta para automatizar el despliegue de modelos en el ámbito de la asignatura Aprendizaje Automático de cuarto curso de Ingeniería Informática. Esta solución, aunque cumple con nuestro principal cometido, que es desplegar modelos de machine learning de forma automática, cuenta con un gran margen de mejora. A continuación destacaremos algunos puntos relevante a la hora de mejorar este sistema MLOps.

En las pruebas que hemos realizado, el clúster se ha desplegado en una máquina local a la que no se puede acceder desde el exterior. Como hemos visto, para paliar esta situación se ha utilizado la herramienta Ngrok. Sin embargo, en un entorno de producción real el clúster de Kubernetes deberá gestionarse en un servidor, ya sea local o en la nube. Como mejora para el futuro se propone desplegar un servidor que gestione el clúster de Kubernetes. Las opciones a explorar en este sentido también son muy amplias, ya que existen numerosas opciones en la nube enfocadas precisamente al mantenimiento de estos clúster encargados de alojar aplicaciones.

Por otro lado, a la hora de definir los trabajos de GitHub Actions, se decidió que la creación de la imagen fuera tarea del servidor virtual de GitHub. Esto supone que, cada vez que se ejecuta el workflow, no solo se tiene que crear la imagen del modelo, sino que también se tienen que instalar una serie de dependencias. Esta máquina virtual no guarda su estado, por lo que podríamos plantear la opción de que el servidor virtual de GitHub solo se encargue de conectarse al servidor que aloja el clúster, de forma que todos los trabajos, desde la creación de la imagen hasta el despliegue, se ejecuten en el servidor real. Esto supondría una mejora en el tiempo que tardamos en desplegar nuestros modelos.

Cabe destacar que en la solución propuesta tan solo se han abordado las partes más esenciales de la funcionalidad propuesta en el capítulo de análisis. Los alumnos podrán subir sus modelos y que estos se desplieguen automáticamente gracias al repositorio de GitHub y a la definición de los flujos de trabajo. El administrador podrá gestionar los modelos desplegados gracias al dashboard que viene integrado en clúster

local de Minikube. Por último, los usuarios, concretamente el profesor de la asignatura, podrá enviar peticiones a los modelos a través de la interfaz Swagger o mediante la herramienta Curl. Sin embargo, todavía quedan por explorar varios puntos de la funcionalidad, como la gestión de los usuarios que tienen acceso al sistema o la consulta de todos los modelos disponibles en el sistema.

Por último, otra punto que no se ha desarrollado en la solución propuesta es la monitorización de los modelos. Como hemos visto, existen bastante herramientas destinadas a ello, por lo que otro camino de mejora sería añadir métodos de monitorización de modelos al sistema. El propio Seldon Core propone la integración con Prometheus, una herramienta para recolectar métricas y gestionar alertas.

En conclusión, durante este trabajo hemos podido ver la importancia de la aplicación del enfoque MLOps para un desarrollo satisfactorio de los proyectos de machine learning. Hemos presentado una solución MLOps que, aunque debe seguir desarrollándose, cumple con la funcionalidad básica para la gestión de los modelos de los alumnos de la asignatura Aprendizaje Automático. Este proyecto podría seguir desarrollándose en el futuro para obtener, finalmente, una aplicación MLOps completa que ayude tanto a los alumnos, que podrán probar sus modelos en un entorno de producción, como al profesor de la asignatura, que podrá evaluar el trabajo de sus alumnos en igualdad de condiciones y sin las complicaciones del despliegue.

---

## 7. Bibliografía

---

- [1] The Kubeflow Authors. Kubeflow documentation, 2022. URL <https://www.kubeflow.org/docs/>.
- [2] BentoML. Bentoml github repository, 2022. URL <https://github.com/bentoml/BentoML>.
- [3] Ernest Chan. Lessons on ml platforms — from netflix, door-dash, spotify, and more. <https://towardsdatascience.com/lessons-on-ml-platforms-from-netflix-door-dash-spotify-and-more-f455400115c7>, 2021.
- [4] The Linux Foundation. Documentación de kubernetes, 2022. URL <https://kubernetes.io/es/docs/home/>.
- [5] Inc. GitHub. Github actions documentation, 2022. URL <https://docs.github.com/es/actions>.
- [6] Docker Inc. Documentación de docker, 2022. URL <https://docs.docker.com/>.
- [7] iterative.ai. Cml documentation, 2022. URL <https://cml.dev/doc>.
- [8] KServe. Kserve github repository, 2022. URL <https://github.com/kserve/kserve>.
- [9] Kumar Chandrakant. Service mesh architecture with istio. <https://www.baeldung.com/ops/istio-service-mesh>, 2021.
- [10] Seldon Technologies Ltd. Seldon core documentation, 2022. URL <https://docs.seldon.io/projects/seldon-core/en/latest/index.html>.
- [11] Matthias Breunig. Why mlops is critical to the future of your business. <https://www.forbes.com/sites/googlecloud/2021/05/19/why-mlops-is-critical-to-the-future-of-your-business/?sh=4d2ee7e14537>, 2021.
- [12] Metaflow. Metaflow documentation, 2022. URL <https://docs.metaflow.org/>.
- [13] Mihindu Ranasinghe. Using docker containers in jobs - github actions. <https://dev.to/mihinduranasinghe/using-docker-containers-in-jobs-github-actions-3eof>, 2020.
- [14] MLflow Project. MLflow documentation, 2022. URL <https://www.mlflow.org/docs/latest/index.html#mlflow-documentation>.
- [15] Red Hat GitHub Actions. Openshift tools installer action. <https://github.com/marketplace/actions/openshift-tools-installer>, 2022.

- [16] Stenac, Clement, Leo Dreyfus-Schmidand, Kenji Lefevre, Nicolas Omont, and Mark Treveil. *Introducing MLOps*. O'Reilly Media, Inc., 2021.
- [17] The Institute for Ethical Machine Learning. Awesome production machine learning. <https://github.com/EthicalML/awesome-production-machine-learning#model-serving-and-monitoring>, 2022.
- [18] Wikipedia contributors. Continuous delivery — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/w/index.php?title=Continuous\\_delivery&oldid=1080945954](https://en.wikipedia.org/w/index.php?title=Continuous_delivery&oldid=1080945954), 2022.
- [19] Wikipedia contributors. Continuous integration — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/w/index.php?title=Continuous\\_integration&oldid=1085884364](https://en.wikipedia.org/w/index.php?title=Continuous_integration&oldid=1085884364), 2022.
- [20] Wikipedia contributors. Devops — Wikipedia, the free encyclopedia. <https://en.wikipedia.org/w/index.php?title=DevOps&oldid=1089933092>, 2022.
- [21] Wikipedia contributors. Kubeflow — Wikipedia, the free encyclopedia. <https://en.wikipedia.org/w/index.php?title=Kubeflow&oldid=1086825923>, 2022.
- [22] Wikipedia contributors. Mlops — Wikipedia, the free encyclopedia. <https://en.wikipedia.org/w/index.php?title=MLOps&oldid=1090024439>, 2022.

---

# A. Anexo. Manual del administrador

---

En este anexo se explicará, a modo de manual, como llevar a cabo las tareas propias del administrador del sistema MLOps. Abarcaremos la instalación de herramientas, la creación del clúster y la gestión de los distintos modelos desplegados.

Todos los pasos aquí explicados se han realizado sobre una máquina con el sistema operativo **Ubuntu 20.04**, por lo que algunos pasos podrían cambiar si se parte de una máquina con otro sistema operativo.

## A.1. Instalación de herramientas

### A.1.1. Docker

Para poder ejecutar nuestros modelos dentro del clúster de Kubernetes vamos a necesitar tener Docker instalado. Para la instalación tenemos que ejecutar los siguientes comandos en la consola de Linux:

```
$ sudo apt update

$ sudo apt install apt-transport-https ca-certificates curl
  software-properties-common

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
  sudo apt-key add -

$ sudo add-apt-repository "deb [arch=amd64]
  https://download.docker.com/linux/ubuntu focal stable"

$ sudo apt update

$ sudo apt install docker-ce
```

### A.1.2. Kubectl

Kubectl es una interfaz de línea de comandos que nos va a permitir ejecutar comandos sobre nuestro clúster de Kubernetes. Para instalarlo tenemos que ejecutar los siguientes comandos:



```
$ curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

$ curl -LO "https://dl.k8s.io/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"

$ echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check

$ sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

### A.1.3. Minikube

Minikube es una herramienta para ejecutar clústeres de Kubernetes de forma local. En la documentación de Seldon Core se utiliza Kind, otra herramienta con una función similar a Minikube. Sin embargo, esta última permite acceder a un dashboard que puede resultar de utilidad para llevar a cabo tareas de administración del clúster. Para instalar Minikube tenemos que ejecutar los siguientes comandos:

```
$ curl -LO https://storage.googleapis.com/minikube/releases/latest
/minikube-linux-amd64

$ sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Una vez instalado, podemos iniciar el clúster con el comando **\$ minikube start**. Después tenemos que configurar kubectl para que utilice este clúster. Esto lo conseguimos configurando el contexto de Kubernetes con el comando **\$ kubectl cluster-info --context minikube**.

### A.1.4. Istio

Una vez que ya hemos creado el clúster, vamos a instalar la malla de servicio Istio. Para ello ejecutamos los siguientes comandos:

```
$ curl -L https://istio.io/downloadIstio | sh -

$ cd istio-1.11.4

$ export PATH=$PWD/bin:$PATH

$ istioctl install --set profile=demo -y
```

A continuación añadiremos una etiqueta al namespace default de Kubernetes para que Istio inyecte proxies automáticamente junto con cualquier cosa que implementemos en ese espacio de nombres. Esto lo conseguimos con el siguiente comando:

```
$ kubectl label namespace default istio-injection=enabled
```

Para que Seldon Core pueda usar las funciones de Istio para administrar el tráfico del clúster, debemos crear un gateway ejecutando el siguiente comando:

```
kubectl apply -f - << END
  apiVersion: networking.istio.io/v1alpha3
  kind: Gateway
  metadata:
    name: seldon-gateway
    namespace: istio-system
  spec:
    selector:
      istio: ingressgateway # use istio default controller
    servers:
      - port:
          number: 80
          name: http
          protocol: HTTP
        hosts:
          - "*"
END
```

### A.1.5. Seldon Core

Antes de instalar el operador de Seldon Core, crearemos el namespace en el que vamos a alojarlo. Para ello utilizamos el comando **\$ kubectl create namespace seldon-system**.

Para instalar Seldon Core, vamos utilizar la herramienta Helm, por lo que antes tendremos que instalarla. Para ello hay que ejecutar los siguientes comandos:

```
$ curl -fsSL -o get_helm.sh
  https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3

$ chmod 700 get_helm.sh

$ ./get_helm.sh
```

Una vez tenemos Helm instalado, podemos ejecutar el siguiente comando para instalar Seldon Core en nuestro clúster de Kubernetes:

```
helm install seldon-core seldon-core-operator \
  --repo https://storage.googleapis.com/seldon-charts \
  --set usageMetrics.enabled=true \
  --set istio.enabled=true \
  --namespace seldon-system
```

Como el clúster de Kubernetes se ejecuta localmente, tendremos que redirigir un puerto de nuestra máquina local a un puerto de clúster para poder acceder a él externamente. Podemos conseguir esto ejecutando el siguiente comando:

```
kubectrl port-forward -n istio-system svc/istio-ingressgateway 8080:80
```

Esto reenviará cualquier tráfico desde el puerto 8080 en la máquina local al puerto 80 dentro del clúster.

## A.2. Dashboard de Kubernetes

Minikube nos permite acceder a una interfaz de usuarios integrada de Kubernetes en la que podemos realizar acciones sobre nuestro clúster. Podremos desplegar aplicaciones en el clúster, solucionar problemas de las aplicaciones desplegadas, manejar los recursos, obtener una visión general de las aplicaciones en ejecución, entre otras acciones.

Para iniciar la interfaz tenemos que ejecutar el comando **\$ minikube dashboard**. Al hacerlo se abrirá el dashboard en nuestro navegador web predeterminado.