UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MASTER

Design and Development of Automatization Pipelines for Machine Learning Projects based on the MLOps Frameworks

Aitor Martín-Romo González
2023

TRABAJO DE FIN DE MASTER

Título:	Design and Development of Automatization Pipelines for Machine Learning Projects based on the MLOps Frameworks
Título (inglés):	Design and Development of Automatization Pipelines for Machine Learning Projects based on the MLOps Frameworks
Autor:	Aitor Martín-Romo González
Tutor:	Carlos Ángel Iglesias Fernández
Ponente:	PONENTE
Departamento:	Departamento de Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	—
Vocal:	—
Secretario:	—
Suplente:	—

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO DE FIN DE MASTER

Design and Development of Automatization Pipelines for Machine Learning Projects based on the MLOps Frameworks

Aitor Martín-Romo González

Febrero 2023

Resumen

Este Trabajo Fin de Máster (TFM) se centra en la evaluación, selección y despliegue de entornos MLOps, con especial atención al análisis, diseño, implementación y validación de sistemas para la gestión de proyectos de Machine Learning. En este estudio se compararán múltiples plataformas y herramientas MLOps disponibles en el mercado, considerando criterios como la automatización del flujo de trabajo, la gestión de modelos, la escalabilidad y la integración continua. Tras un análisis exhaustivo, se seleccionará la opción más adecuada y se implementará en un caso de estudio real.

Este TFM pretende ofrecer una hoja de ruta detallada para la implantación con éxito de proyectos de aprendizaje automático, destacando la importancia de los aspectos de desarrollo de todo el proceso. Los resultados de este estudio permitirán a las organizaciones optimizar sus operaciones de Machine Learning y mejorar la eficiencia en el despliegue de modelos, contribuyendo a un avance significativo en la gestión de proyectos de inteligencia artificial.

Palabras clave:

Abstract

This Master's Thesis (TFM) focuses on the evaluation, selection and deployment of MLOps environments, with a focus on the analysis, design, implementation and validation of systems for the management of Machine Learning projects. In this study, multiple MLOps platforms and tools available in the market will be compared, considering criteria such as workflow automation, model management, scalability and continuous integration. After a thorough analysis, the most suitable option will be selected and implemented in a real case study.

This TFM aims to provide a detailed roadmap for the successful implementation of machine learning projects, highlighting the importance of the development aspects of the whole process. The results of this study will enable organisations to optimise their Machine Learning operations and improve efficiency in the deployment of models, contributing to a significant advance in the management of artificial intelligence projects.

Keywords:

Agradecimientos

A Gauss

Contents

Resumen **VII**

Abstract **IX**

Agradecimientos **XI**

Contents **XIII**

List of Figures **XV**

1 Introduction **1**

1.1 Context 2

1.2 Project goals 3

1.3 Structure of this document 4

2 State of Art **5**

2.1 First software development: structured programming 6

2.2 Waterfall methodology 6

2.3 Agile Methodologies 7

2.4 Foundations of DevOps 8

2.5 MLOps 9

2.5.1 Definitions and challenges 9

2.5.2 Principles and roles of the machine learning 10

2.5.3 Life cycle of data 12

2.5.4 Madurity levels of machine learning 13

2.6 MLOps frameworks 14

2.6.1 TensorFlow 15

2.6.2 MLFlow 16

2.6.3 KubeFlow 17

2.6.4 Tempo Framework 18

3 Enabling Technolgies **21**
3.1 Machine Learning Technologies 22
3.1.1 TensorFlow 22

4 Architecture and Methodology **23**
4.1 Methodology 24
4.2 Architecture 25

5 Case study **27**
5.1 Techniques 28
5.2 Datasets 28
5.3 Results 28

6 Conclusions **29**
6.1 Achieved Goals 30
6.2 Conclusion 31

A Example **33**

Bibliography **34**

List of Figures

2.1 Comparison of waterfall and agile methodologies 7
2.2 MLOps birth 9
2.3 Comparison of DevOps and MLOps 10
2.4 Comparison of DevOps and MLOps 12
2.5 Google madurity levels 14
2.6 Microsoft madurity levels 14
2.7 General eschema of tempo framework 19

CHAPTER 1

Introduction

This chapter is going to introduce the context of the project, including a brief overview of all the different parts that will be discussed in the project. It will also break down a series of objectives to be carried out during the realization of the project. Moreover, it will introduce the structure of the document with an overview of each chapter.

1.1 Context

In the recent years, the number of projects using machine learning has increased exponentially, as has the amount companies are investing in this technology. This growth carries out with it a bunch of problems since the first models until now, such as the incorporation of machine learning models in production. Until 2022, up to Deborah Leff, former CTO for data science and AI at IBM, 87% of the data science projects never make it into production [15] and among the 90% of companies that have made some investment in AI, fewer than 2 out of 5 report business gains from AI, improving this number to 3 out of 5 when we include companies that have made significant investments in AI [12]. Those are the reasons why only a small percentage of the ML projects manage to reach production, being essential to find out what are the problems which come across since such an extraordinary inversion from the companies should never be wasted.

1.2 Project goals

Describe the different goals of the project

1.3 Structure of this document

The remaining of this document is structured as follows:

Chapter 2 ... (Chapter 1 is the Introduction)

CHAPTER 2

State of Art

In order to comprehend the current situation of the software industry, we must understand the previous steps of the software path.

2.1 First software development: structured programming

The first software developments are about the 40s and 50s, when the first computer machines came out. Such was the rise of these new technologies that software's developers had to give an extra twist to make them useful. So, they started developing software using Structured Programming.

Structured Programming is a trend that was born to make the life of the developers easier. It was not until 1966, when Böhm and Jacopini launched the structured program theorem, which says that any program could be wrote using just 3 instructions, when its consolidation began [16]. In 1968, Edsger Dijstra published a well-known article, Go To Statement Considered Harmful [9], claiming the use of this new concept and the banish of the Goto sentence. Ending the consolidation of the structured programming.

As it has been said before, it is based in the 3 basic structures: sequence, selection or conditional and iteration. Those three basics made easier to understand the codes, with a more clear structure, better to optimize the testing and debugging and the maintenance expenses were reduced [16].

2.2 Waterfall methodology

At the beginning of the 70s the projects were complex enough having a long list of requirements, also needing a lot of documentation, alongside a proper design's planning of the solution. Aiming to solve these new challenges, Waterfall methodology was born.

Based on a correct requirement' definition, due they must be unchanged during all the process, it is a linear process of project management. Each step on the procedure cannot begin unless the previous phase is finished, and once finished, it is terminal, since Waterfall management does not allow you to return to a previous phase [14]. Normally, waterfall methodology varies somewhat depending on the source, but they generally include:

- Requirements gathering and documentation.
- System design
- Implementation
- Testing
- Delivery/Deployment
- Maintenance

2.3 Agile Methodologies

Due to the slowness and the delays caused by the traditional way of working, which used waterfall methodologies, the software industry ideated the agile methologies. The previous projects were based in fixed requirements, not able to change them once the process started, and the big efforts that were meant to suppose if a change was made drove to not-as-high-as expected quality projects.

In 2001, the Agile Manifest was created by the principal CEOs of the software industry in Utah [8]. The Agile manifest was based in four keys:

- Iterations and individuals above processes and tools
- Functional products above exhaustive documentations.
- Partnership above deal negotiations.
- Change with the problem above and strict plan.
- Delivery/Deployment
- Maintenance

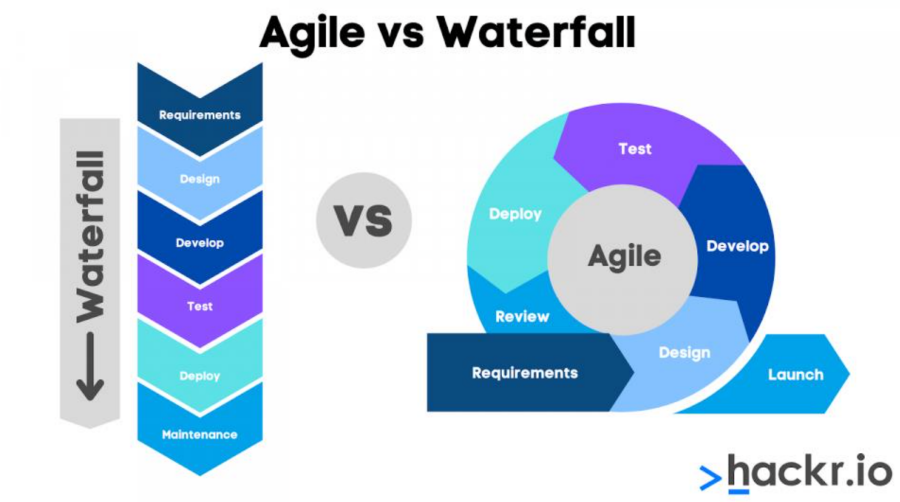

Figure 2.1: Comparison of waterfall and agile methodologies

The main advantages of the agile methodologies that will make it be the base of a big amount of DevOps processes are:

- Ease and reduction of the process overload: being able to adapt at every stage of the development process makes easier to achieve what is expected in every milestone, with no extra effort at the end of it.

- Better quality of the product: at every iteration at least a minimum functionality, but at the end, an improvement from the previous ones. Moreover, the customer is at every moment involved in the development process, being able to ask for changes depending on the market realities.
- Improvement in the foresees through a better management of the risk: The agile methodologies were designed to respond the possible changes and problems that merge during the life cycle of the project.
- The customer is involved at every stage of the development so they could give feedback at every point, making a stronger relationship between developers and customer and an improvement in the satisfaction.

2.4 Foundations of DevOps

The previous both methodologies have the same objectives, deliver production-ready software products. The problems came with the gap between the developer teams and the operation teams, reaching even point of isolation between them, competing against each other's. Both teams had different KPIs, different squad leaders, objetives. So, a concept called DevOps emerged in the years 2007-2009 [11] trying to solve all the differences. DevOps is more than a pure methodology and rather represents a paradigm addressing social and technical issues in organizations engaged in software development [10].

During the life cycle of the product, DevOps uses agile methodologies by definition. Also is based in the automation of processes, it is managed through the continuous integration, continuous delivery and continuous deployment (CI/CD). It allows to deliver fast and reliable solutions too. Moreover, it is designed to facilitate continuous testing and quality assurance, and thanks to other applications and tools, monitoring, logging and feedback loops. The most important assets of the DevOps methodology are [10]:

- Workflows and repositories, also called source code management: Tools such as GitHub, BitBucket or GitLab.
- Monitoring and Logging (e.g., Prometheis, Logstash).
- Build process (e.g., Maven).
- Continuos integration (e.g., Jenkins, GitLab CI)
- Deployment automation (e.g., Kubernetes, Docker, Ansible)

Nowdays, the main Cloud providers have already-built solutions for DevOps tooling, reducing the time needed to start giving value to a project.

2.5 MLOps

2.5.1 Definitions and challenges

Therefore, summing up everything said before and at gross mode, MLOps is the application of the DevOps methodologies alongside the use of machine learning in a production environment. In a more technical way, MLOps is the standardization and streamlining of machine learning life cycle management [13]. For most organizations and companies, the process of the machine learning is relatively new and the number of projects in productions is not big enough yet to accommodate them to an automation environment, which is where it becomes more critical. For those that have done it, the main challenges they face during the life cycle of the data are:

- Changing environment: data and the business needs shift constantly, is required to be sure that the model in productions aligns with the expectations and if it satisfies the original problem and goals.

- Misleading communication: despite of being in the same company or share the same goals, not everyone shares the same tools, procedures or skills.

With the gathered facts explained before, it was easy to come out with a solution truly related with the DevOps models, MLOps, which is quite similar but not identical since the software development used in DevOps is almost static, almost because some companies can iterate changes during the software life cycle, against the continuous changing data of Machine Learning. Machine learning models are always learning and responding to new environment it is in, also including both data and code.

So, the final model of the MLOps was born from the development and operations of the DevOps and the data engineering.

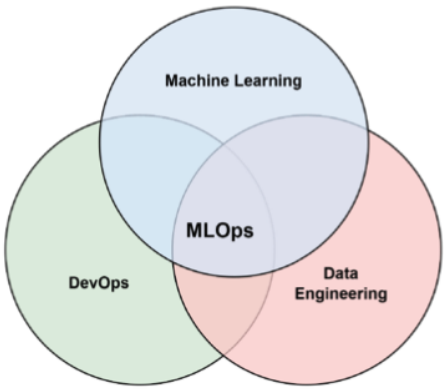

Figure 2.2: MLOps birth

The greatest similarity between MLOps and DevOps methodologies lies in the concepts of Continuous Integration and Continuous Delivery, which allow software deliveries to occur with a high frequency, with reliable results. In most companies that use Machine Learning, they develop the different models and put them into production manually, without incurring MLOps. This also brings with it the problem that machine learning has no return of investment until it can be used. Therefore, all efforts must focus on the steps that follow the development of the model itself, specifically on the interfaces between the ML response and the infrastructure where it will be implemented [13].

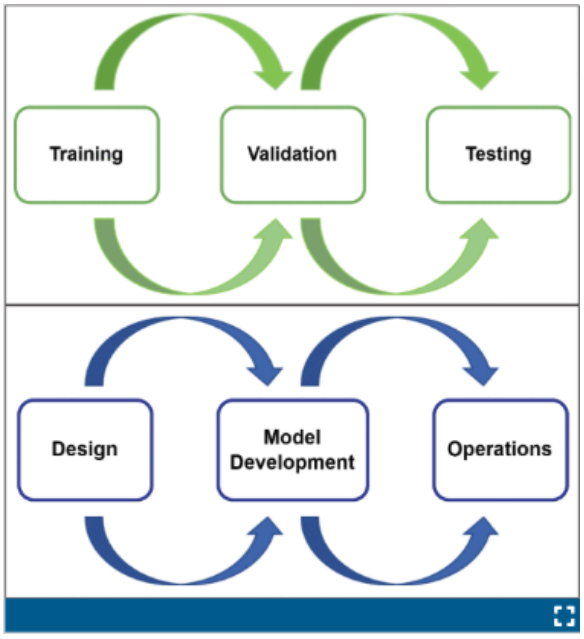

Figure 2.3: Comparison of DevOps and MLOps

2.5.2 Principles and roles of the machine learning

As with any methodology, MLOps is also governed by a set of principles. These are not fixed, but any process that aims to end up with a machine learning model in production will have to be governed by certain fundamentals such as the following:

- CI/CD automation: this process carries out the building, testing, shipping and deployment of software through continuous integration, continuous delivery and continuous deployment. As previously mentioned, this allows for early identification of process failures and continuous product value and increased productivity.

- Reproductivity: the ability to replicate the same processes obtaining the same results in different environments for the same inputs.

- Workflow orchestration: coordination of the different machine learning tasks in a pipeline according to acyclic graph directives.
- Versioning: different versions of data, models and code allow for greater control over regulatory compliance and audits.
- Continuous monitoring: by continuously monitoring the data, the model, the code, or the infrastructure itself, errors can be detected and/or changes can be made according to previous results in a more efficient and less costly way.
- Feedback loops: this type of loop is necessary to manage the relationship between the different stages of quality assessment and the engineering or development processes.
- Continuous re-training of the different models: as each re-training is based on new data, it is a consequence of the previous processes of continuous monitoring, feedback loops and workflow orchestration. One aspect to be managed is the expenditure to be allocated to this stage, as a higher frequency of re-training entails a much higher cost.

Continuing with the roles needed for an MLOps process, these are based on agile methodologies. As in any software development process, a good definition of the participants in the process is fundamental to design, manage, automate and operate any machine learning system.

- Business stakeholder: in charge of defining the business objectives is also in charge of taking care of the communication between the team and the client.
- Solutions architect: defines the architecture and technologies that will be used.
- Data scientist: converts business problems and requirements into machine learning problems. At a technical level, he/she oversees model engineering.
- Data engineer: designs and implements data pipelines and engineering features.
- Software engineer: converts the ML problem given by the data scientist into a well-engineered product by applying software framework design and best practices.
- DevOps Engineer: strives to link development and management processes under CI/CD methodology, ML workflow orchestration, model deployment to production and monitoring.
- combination of all the above roles, cross-functionally. Manages the ML infrastructure and ML automation flows.

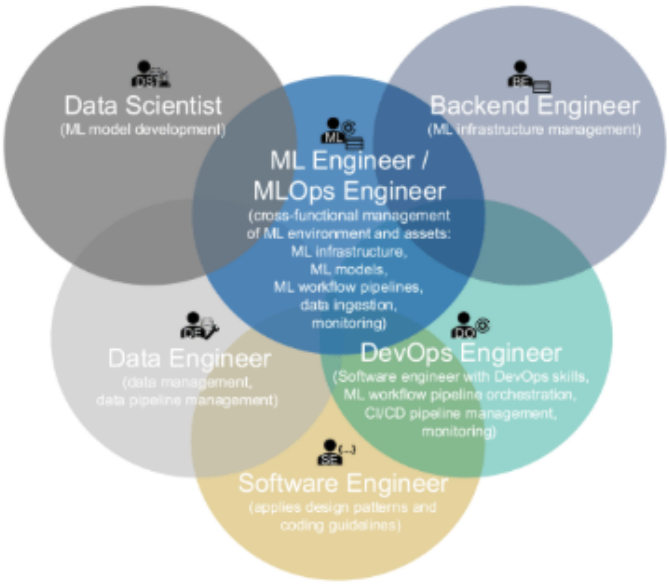

Figure 2.4: Comparison of DevOps and MLOps

2.5.3 Life cycle of data

In general, all AI projects, whether machine learning or deep learning, share common phases ranging from business understanding to model deployment. They also include stages such as data transformation and explanatory analysis. It is worth noting that this process is not linear, but circular or iterative, as once the model is deployed, it needs to be monitored and revised to continuously adapt to the needs of the business. This process is known as the Cross Industry Standard Process for Data Mining (CRISP- DM) and is the most widely used analytical model [7]. Both the Google and Microsoft representations are intended to illustrate that the process is iterative and that at all times it is essential to maintain a clear understanding of the desired outcomes and objectives of each phase in order to carry out the project as efficiently as possible:

1. Understanding the business and the data: The first step before development begins involves understanding the use case, the sector you will be working in and the specific problems you intend to address. It is also essential to know all the details about the data required, including its location, format and how it can be used effectively. This understanding is essential to focus the objectives of the project and to ensure that you have the data you need to meet those objectives. In addition, knowing the format and location of the data is crucial to its efficient extraction and exploitation.

2. Data ingestion: The data ingestion phase involves obtaining the data, and this varies significantly depending on the sources and frequency of data collection. In this phase as in the previous one, a careful assessment of the amount of data and the type of

temporal architecture is vital to avoid problems related to system capacity as the volume of data increases. Accurate volume estimation is crucial, as many data science projects face challenges in moving into production due to underestimating the volume of data or getting too little data compared to the amount generated.

3. Data preparation and cleaning: Although this phase is related to the previous ones, it stands out because it is often one of the most complex stages and, in most projects, the most extensive. Data quality is essential for AI models to work properly, so data preparation and data cleansing are essential. Although automation and standardization of this task is often sought, human supervision is preferable, as data quality is crucial for better results in the modelling phase.

4. Exploratory data analysis: In this stage, also known as data visualization, clean and prepared data is interpreted and visualized to make informed decisions and understand it in the best possible way. Visualization and representation of the value of the data through reporting tools, graphs and charts are essential in a data project and allow critical questions to be addressed to advance development.

5. Modelling and model evaluation: Modelling is considered the core of data analysis. A model takes the prepared data as input and provides the desired data as output. At this stage, important decisions are made, such as the choice of the appropriate model and the selection of hyperparameters that optimize its performance. Performance is measured by various metrics, which vary from model to model, in order to evaluate the model and obtain the best possible result.

6. Deployment of the model: Deployment is one of the critical parts of development and is closely related to DevOps. From deployment onwards, it is necessary to continuously monitor to detect problems and adapt to changing business needs. Despite the importance of MLOps techniques in previous stages, it is at this point where they become of significant value, as they allow problems to be detected efficiently, corrected, and new models deployed quickly, achieving an effective system with the best possible results and long-term sustainability.

2.5.4 Madurity levels of machine learning

The level of automation within a MLOps system can be categorized into corresponding levels. In spite of the fact there is no accepted maturity model by the community, the two biggest hiperscallers companies, Microsoft and Google, have raised with two proposals, one each one. The model from Google consists in three levels [4] :

- MLOps Level 0: No automation at all, everything is done manually.
- MLOps Level 1: Automation of the ML pipelines.

– MLOps level 2: Automation of CI/CD pipelines.machine learning

Figure 2.5: Google madurity levels

The Microsofts' model has 5 levels instead [5]

– Level 0: No MLOps.

– Level 1: DevOps without MLOps.

– Level 2: Automated training.

– Level 3: Automated model deployment.

– Level 4: Fully automated MLOps Operations.

Figure 2.6: Microsoft madurity levels

2.6 MLOps frameworks

An MLOps framework can be defined as a conceptual and technological structure, i.e., a set of tools, processes and best practices designed to manage and automate the lifecycle

of machine learning projects, from model development and training to implementation and maintenance in production environments. An MLOps framework generally includes components and features such as:

- Version management: To track changes to code, data and models.
- Workflow automation: To automate tasks such as data collection and preprocessing, model training, and production deployment.
- Environment management: To ensure that development, test and production environments are consistent.
- Monitoring and logging: To track the performance of models in production and ensure their proper operation.
- Testing and validation: To verify the quality and performance of the models before deploying them in production.
- Deployment and orchestration: To implement models in production environments in an efficient and scalable way.
- Model management: To track model versions, manage their lifecycle and enable the update and deployment of new versions.
- Security and compliance: To ensure data and model security and comply with regulations and privacy policies.
- Collaboration and communication: To foster collaboration between data science, development and operations teams.

Examples of popular MLOps frameworks include MLflow, Kubeflow, TFX (TensorFlow Extended), Metaflow and others. These frameworks provide a structure that facilitates the implementation of MLOps practices in machine learning projects, helping to ensure reliability, scalability and efficiency in managing models in production. Some examples, alongside the framework finally used, will be explained below.

2.6.1 TensorFlow

TensorFlow is an open source library developed by Google for machine learning and deep learning. Some of its main features and key concepts are presented below [1]:

- Computational Graphs: TensorFlow represents computations as a directed graph. The nodes of the graph represent mathematical operations, and the edges represent

the data (tensors) that flow between operations. This allows for efficient optimization and parallelization of computations.

- Deep Learning: TensorFlow is primarily used in deep learning, a subarea of machine learning that focuses on deep neural networks to solve complex tasks such as computer vision, natural language processing and more.
- High-Level API: TensorFlow offers high-level APIs, such as Keras, that facilitate the creation and training of deep learning models. Keras provides a simple and easy-to-use interface for defining and training neural networks.
- Flexibility: TensorFlow is highly flexible and can be used on a variety of platforms and devices, including CPUs, GPUs and TPUs (Tensor Processing Units), making it suitable for machine learning tasks in diverse environments.
- Extensive Ecosystem: TensorFlow has an extensive ecosystem of tools and extensions for specific machine learning and deep learning tasks. This includes TensorFlow Serving for deploying models in production, TensorFlow Lite for mobile and embedded devices, and TensorFlow.js for web applications.
- Autodifferentiation: TensorFlow enables automatic differentiation, which is crucial in optimizing machine learning models by gradient descent. This means that TensorFlow can calculate gradients automatically to adjust model parameters.
- Pre-trained Models: TensorFlow provides access to pre-trained models on large datasets, which can accelerate the development of machine learning applications by enabling transfer of learning.
- Active Community: TensorFlow has an active developer community and extensive documentation, which facilitates problem solving and adoption of new features.
- Interoperability: TensorFlow is compatible with several programming languages, including Python, C++, and more, making it accessible to a wide audience of developers.
- Distribution: TensorFlow is capable of distributing training and prediction tasks across clusters of machines, which is essential for training models on large datasets.

2.6.2 MLFlow

MLflow is an open source platform developed by Databricks to manage the lifecycle of machine learning projects. It allows teams of developers and data scientists to manage experiments, keep track of models, collaborate and track implementations efficiently. Here are some of the key features of MLflow [3]:

- Experiment Management: MLflow allows users to keep track of machine learning experiments. You can organize and compare different model runs to evaluate their performance and make data-driven decisions.
- Model Registration: MLflow allows you to register and version trained models along with their metadata and parameters. This facilitates model reuse and deployment in different environments.
- Version Control: MLflow provides a version control system for models and experiments, which helps maintain a history of all changes made to code and models.
- Model Packaging: Models registered in MLflow can be packaged in standard formats (e.g., in Docker containers) for easy deployment and deployment in different environments, such as the cloud or embedded devices.
- Parameter and Metric Tracking: MLflow enables tracking and visualization of metrics, parameters and artifacts associated with each run of an experiment, making it easier to understand how a model behaves in different configurations.
- Integration with Machine Learning Libraries: MLflow integrates seamlessly with popular machine learning libraries such as scikit-learn, TensorFlow, PyTorch, XGBoost and others, making it easy to record models and training results.
- Model Collaboration and Sharing: Teams can effectively collaborate on machine learning projects and share registered models with other team members.
- Model Deployment in the Cloud: MLflow makes it easy to deploy models on popular cloud services such as Azure ML, AWS SageMaker, Google AI Platform and others.
- APIs for Different Languages: MLflow offers APIs for Python, R and REST, providing flexibility in the choice of programming languages to work with MLflow.

2.6.3 KubeFlow

Kubeflow is an open source platform specifically designed to facilitate the development, deployment and management of machine learning applications in Kubernetes environments. Kubernetes is a widely used container orchestration system, and Kubeflow extends its capabilities to meet the needs of teams working in machine learning. Here are some of the key features of Kubeflow [2]:

- Model Orchestration: Kubeflow enables orchestration of machine learning workflows, making it easy to create, train and deploy models in a scalable and reproducible manner.

- Unified Development Environment: provides a unified environment for developers and data scientists, facilitating machine learning project collaboration and resource management.
- Data Management: Provides tools for data management, including data access, data preparation and feature engineering.
- Experimentation and Model Tracking: Kubeflow allows keeping a record of experiments and models, which facilitates the comparison of results and informed decision making.
- Model Deployment: Facilitates the deployment of models in production environments through its integration with Kubernetes, enabling scalability and high availability.
- Elasticity: Kubeflow leverages the scalability of Kubernetes to adapt resources as needed, allowing it to handle varying machine learning workloads.
- Extensibility: Kubeflow's architecture is modular and extensible, which means that additional components and functionality can be added according to the specific needs of the project.
- Support for Diverse Machine Learning Libraries: Kubeflow is compatible with a variety of machine learning libraries, such as TensorFlow, PyTorch, scikit-learn and others, which provides flexibility in the choice of technologies.
- Monitoring and Visualization: Provides tools to monitor and visualize metrics and results of experiments and models.
- Web User Interface: Kubeflow offers an intuitive web user interface that allows users to manage and control workflows, experiments and models efficiently.

2.6.4 Tempo Framework

Tempo is an open source framework for Python that allows testing and development of machine learning pipelines, which can be created and tested locally or deployed via Seldon in production. Among the main features of tempo are [6]:

- Optimisation of model packages for shorter execution time on servers.
- Ease of orchestration of process steps.
- Support for any custom Python component.

– Test locally and deploy in production, with the possibility of following GitOps workflows.

The general outline of the workflow of this framework is as follows:

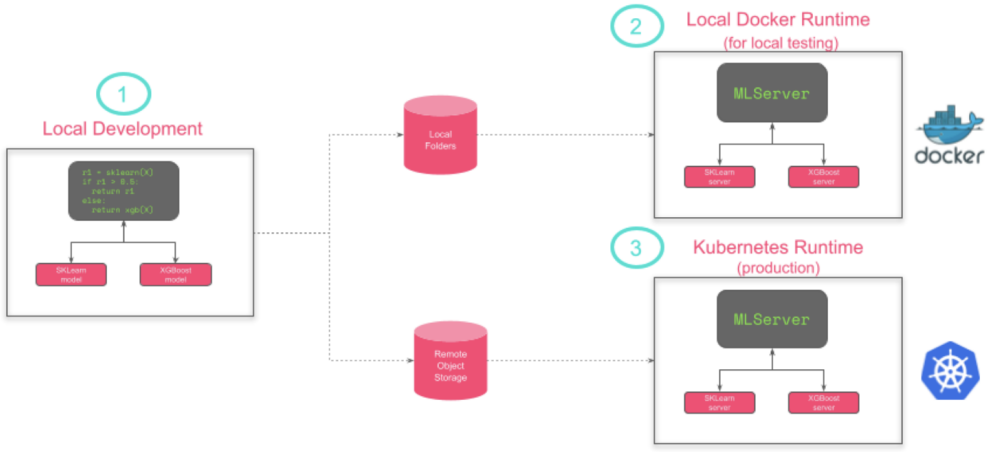

Figure 2.7: General eschema of tempo framework

CHAPTER 3

Enabling Technolgies

This chapter offers a brief review of the main technologies that have made possible this project, as well as some of the related published works.

3.1 Machine Learning Technologies

3.1.1 TensorFlow

CHAPTER 4

Architecture and Methodology

This chapter presents the methodology used in this work. It describes the overall architecture of the project, with the connections between the different components involved on the development of the project.

4.1 Methodology

4.2 Architecture

CHAPTER 5

Case study

In this chapter we are going to describe a selected use case. This section will discuss the two use case: Sentiment and Emotion Analysis applying Deep Learning techniques, which will include the datasets used, the first steps with the neural networks and the use of more advance techniques.

5.1 Techniques

5.2 Datasets

5.3 Results

...

CHAPTER 6

Conclusions

This chapter will describe the achieved goals done by the master thesis following some the key points developed in the project.

6.1 Achieved Goals

The achieved goals for this project are the following ones:

- ...

6.2 Conclusion

The project has fulfilled the proposed objectives, but we must refer to each one of these objectives in to draw a general conclusion.

APPENDIX A

Example

This is the description of the appendix.

Bibliography

[1] Crea modelos de aprendizaje automático de nivel de producción con tensorflow.

[2] Kubeflow: The machine learning toolkit for kubernetes.

[3] Mlflow: A platform for machine learning lifecycle.

[4] Mlops: canalizaciones de automatización y entrega continua en el aprendizaje automático. *Google*.

[5] Modelos de madurez de operaciones de machine learning. *Microsoft*.

[6] Tempo framework: Tempo: The mlops software development kit.

[7] La metodología para poner en orden los proyectos. *Sngular*, 2020.

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

[9] Edgar Dijkstra. Go to stament considered harmful. 1968.

[10] Dominik Kreuzberger, Niklas Kühl, and Sebastian Hirschl. Machine learning operations (mlops): Overview, definition and architecture. *IEEE Xplore*, 2023.

[11] Steve Mezak. The origins of devops: What's in a name? *DevOps Journal*, 2018.

[12] Sam Ransbotham, Shervin Khodabandeh, Ronny Fehling, Burt Lafountain, and David Kiron. Winning with ai. *MITSloan*, 2019.

[13] Mark Treveil, Nicolas Omont, Clément Stenca, Kenji Lefevre, and Du Plan. Introducing mlops. *IEEE Xplore*, 2022.

[14] Unknown. What the waterfall project management methodology can (and can't) do for you. *LucidChart*, 2023.

[15] VBStaff. Why do 87 *VentureBeat*, 2019.

[16] Ingeniería y tecnología. ¿qué es la programación estructurada? *UNIR*, 2022.