ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

MLOps - Standardizing the Machine Learning Workflow

Thesis on
BIG DATA

Relatore
Dott. Enrico Gallinucci

Correlatore
Dott. Alessandro Bianchi

Candidato
Enrico Salvucci

I Sessione di Laurea
Anno Accademico 2020-2021

Abstract

MLOps is a very recent approach aimed at reducing the time to get a Machine Learning model in production; this methodology inherits its main features from DevOps and applies them to Machine Learning, by adding more features specific for Data Analysis. This thesis, which is the result of the internship at Data Reply, is aimed at studying this new approach and exploring different tools to build an MLOps architecture; another goal is to use these tools to implement an MLOps architecture (by using preferably Open Source software). This study provides a deep analysis of MLOps features, also compared to DevOps; furthermore, an in-depth survey on the tools, available in the market to build an MLOps architecture, is offered by focusing on Open Source tools. The reference architecture, designed adopting an exploratory approach, is implemented through MLFlow, Kubeflow, BentoML and deployed by using Google Cloud Platform; furthermore, the architecture is compared to different use cases of companies that have recently started adopting MLOps.

MLOps is rapidly evolving and maturing, for these reasons many companies are starting to adopt this methodology. Based on the study conducted with this thesis, companies dealing with Machine Learning should consider adopting MLOps. This thesis can be a starting point to explore MLOps both theoretically and practically (also by relying on the implemented reference architecture and its code).

Acknowledgements

Throughout the writing of this dissertation I have received a great deal of support and assistance.

First of all, I would like to thank my supervisor, Prof. Enrico Gallinucci, for the thoroughness with which he took care of me both during the implementation of the reference architecture and during the writing of this thesis.

I would like to thank my colleagues from my internship at Data Reply, especially Alessandro and Michele, for assisting me with great willingness.

I also would like to acknowledge the MLOps Community for the knowledge they shared, which allowed me to deepen various topics in this thesis.

Last but not least, I really want to thank all my closest friends who supported me both during my university career and in the writing of my thesis.

Contents

Abstract **iii**

Introduction **1**

1 MLOps **3**

1.1 DevOps 4

1.1.1 Influence on MLOps 4

1.1.2 MLOps, DataOps, ModelOps and AIOps 5

1.2 The Machine Learning lifecycle 6

1.2.1 A process, not only a product 8

1.2.2 People involved in the Machine Learning lifecycle 9

1.3 State of the Art and the need for MLOps 11

1.3.1 Technical Debt in Machine Learning systems and anti-patterns 12

1.3.2 Challenges 15

1.4 MLOps features 15

1.4.1 Continuous 16

1.4.2 Reproducibility 17

1.4.3 Versioning and Experiment Tracking 17

1.4.4 Testing 18

1.4.5 Monitoring 19

1.4.6 Modularity 19

1.4.7 Automation 19

1.4.8 Workflow Pipeline Design Pattern 22

2 A survey of technologies for MLOps **25**

2.1 Open Source technologies 25

2.1.1 Environment/Containerization 26

2.1.2 Experiments tracking 26

2.1.3 Pipeline Orchestration 30

2.1.4 CI/CD 32

2.1.5 Feature Store 34

2.1.6 Serving 35

3 Design and implementation of a reference architecture for MLOps 37

3.1 Introducing an MLOps architecture in the business process 38

3.2 The reference architecture 39

3.3 Deployment of the architecture 44

3.4 The training and the prediction pipelines in detail 47

3.5 Code publication 51

4 MLOps use cases and scenarios 53

5 Conclusions 61

Glossary 65

List of Figures

1.1 DevOps lifecycle 5
1.2 Components of a Machine Learning system [1] 6
1.3 Machine Learning lifecycle [2] 7
1.4 Trends for MLOps searches, January 2017 - May 2021 11
1.5 Machine Learning models life time [3] 12
1.6 Level 2: CI/CD and automated ML pipeline [4] 22

2.1 Example of different runs of a model training and the tracked parameters and metrics 27
2.2 Example of the parameters recorded in a single run of the model training (with also the the experiment duration, the date and the user who executed it). 28
2.3 Example of different versions of a model trained through MLFlow. The models can also be compared with each other. 29
2.4 Kubeflow components usage (emerged from the survey conducted by the Kubeflow community in march 2021) [3] 31
2.5 Hopsworks architecture [5] 34
2.6 Comparison between the most used Feature Stores [6] 35

3.1 The designed and implemented MLOps architecture [7]. 39
3.2 The DAG of the training pipeline on Kubeflow. 40
3.3 The DAG of the prediction pipeline on Kubeflow. 40
3.4 Successful runs of the model training experiments, tracked through MLFlow. 41
3.5 Runs of the training and predictions pipelines in Kubeflow. 42
3.6 OpenAPI specifications of the BentoML service. 43
3.7 Successful response of a prediction through the BentoML ReST API. 44
3.8 The sequence of the steps performed in the CI/CD flow. 45
3.9 Runs on Google Cloud Build, triggered by a commit or a merge in the code repository. 46
3.10 Example of substitution variables in Google Cloud Build. 46

3.11 The DAG of the training pipeline on Kubeflow, triggered by Google Cloud Functions as consequence of a file upload (or update), while running the model traininig components. 47
3.12 The DAG of the prediction pipeline on Kubeflow while running both the load-data and remove-header components. 47
3.13 The logs of Google Cloud Functions when a new file in the bucket is uploaded (or updated). 50

4.1 Stack of the technologies included in Metaflow [8]. 55
4.2 CI/CD architecture of Itau Unibanco [9]. 56
4.3 MLOps tools stack used by GreenStream [10]. 57
4.4 Architecture of Uber Michelangelo [11]. 58
4.5 MLOps architecture of H&M [12]. 59

Listings

3.1 The definition of the data-ingestion Kubeflow component 48
3.2 The kfp-cli command used to run programmatically the Kubeflow pipeline 49

Introduction

MLOps is a very recent approach aimed at reducing the time to get a Machine Learning model in production. A model may employ many months, or even an entire year, to cover all its end-to-end process. Covid-19 pandemic, for example, has disrupted many supply chains, whose models were not updated frequently enough to handle the change in the data; these changes were mostly caused by the different lockdowns in the world. In addition to the model building and training, the end-to-end Machine Learning process involves many steps; MLOps allows data scientists to focus on all of them. Moreover, MLOps inherits its main features from DevOps and applies them to Machine Learning. Due to the differences among traditional software and Machine Learning models MLOps includes other characteristics such as Continuous Training and Continuous Monitoring; tracking and versioning the experiments performed to build a model are specific features of MLOps as well. By adopting MLOps companies can manage their models with flexibility and update them quickly and easily. MLOps' benefits can really improve the quality of the development of a model and can significantly reduce the time to get the model itself into production.

This thesis is the result of the internship at Data Reply and its purpose is to deeply investigate MLOps methodology from both theoretical and practical perspectives. Given its very recent nature, companies are not yet embracing the idea of bringing the DevOps methodologies into Machine Learning processes. The requirements that Data Reply had for the internship were to acquire the know-how about the state-of-the-art of MLOps, to understand which technological stacks can be set up for an MLOps project and, also, to build and test a reference architecture to serve as a template for future projects. This thesis starts by comparing DevOps and MLOps; thereafter it provides an overview of the state-of-the-art of this very recent approach. A deep analysis of MLOps' features will be also provided. According to the specific requirements of each company, which employs this new methodology, many tools can be adopted to build an MLOps architecture. A deep survey on the main technologies will be presented, by focusing on the Open Source ones; some of the discussed tools have been employed to build the reference architecture for this thesis. The architecture is designed according to an exploratory approach and is implemented by using three Open Source tools: MLFlow, Kubeflow and BentoML; the whole architecture is deployed through Google Cloud Platform. Some considerations on introducing MLOps in a company will be offered; afterward different use cases about how various companies adopt MLOps will be discussed and their architecture will be also compared to the reference architecture of this thesis. Some of the mentioned companies are AstraZeneca, Netflix, Uber and H&M.

Chapter 1 examines MLOps features, including its differences with DevOps and its state-of-the-art. Chapter 2 explains the main technologies to implement an MLOps infrastructure, by focusing on the Open Source ones. Chapter 3 firstly offers some considerations on introducing MLOps in a company; moreover, it describes the implementation of the reference architecture of this thesis. Chapter 4 explains different use cases about how various companies adopt MLOps and the previously mentioned example about Covid-19 as well.

MLOps is rapidly evolving and maturing, for these reasons many companies are starting to adopt this methodology. Based on the study conducted with this thesis, companies dealing with Machine Learning should consider adopting MLOps. This thesis can be a starting point to explore MLOps both theoretically and practically (also by relying on the implemented reference architecture and its code).

Chapter 1

MLOps

A typical pain point in Machine Learning is the large amount of time to get a model into production. *MLOps* is *"an approach in which a cross-functional team produces Machine Learning* [1] *applications; these are based on code, data, and models in small and safe increments that can be reproduced and reliably released at any time, in short adaptation cycles"* [13]. Due to the large amount of time required to get a model into production and due to the small lifetime of the model itself in production, the full potential of Machine Learning is not currently being reached. The delay between initiating the data science project and deploying the model (so it can make predictions) often leads to having a model in production that no longer conforms to real-world data. One of the main goals for MLOps is to let companies reduce the time to deploy a model and get it into production faster; MLOps is also aimed at providing an approach to simplify and standardize the Machine Learning lifecycle. These objectives require organizations to address different Technical Debts, reduce the gap between data scientists and operational teams and adopt a new approach to develop Machine Learning systems.

The term "MLOps" is strictly related to the DevOps approach. As the name itself suggests, MLOps inherits its main principles from DevOps; yet deploying software code is deeply different from deploying Machine Learning models into production: code is static, data changes constantly. The word "MLOps" has been introduced for the first time by Kaz Sato, Staff Developer Advocate at Google Cloud. He thought that *"DevOps is all about unifying the development and operations (Dev and Ops), we can use the same concepts on the Machine Learning based systems"* [14].

[1]For the sake of consistency, the term *Machine Learning* (ML), is used here in a broadly fashion; MLOps is not referred only to Machine Learning but to all the data science fields (other terms can be used, see Section 1.1.2).

1.1 DevOps

DevOps is a cultural movement, a way of thinking and working, which supports intentional processes that accelerate the rate by which business value is obtained. It has its roots in Agile software development principles and could be considered an extension of them. DevOps is especially focused on the first principle of the *Agile Manifesto*: *"Individuals and interactions over process and tools"* [15]. As DevOps encourages critical thinking about tools it considers them a value, but it does not mandate or require any specific one. It is also crucial to underline that effective tool usage is necessary for a successful DevOps transformation but not sufficient. DevOps emphasizes that interactions and collaborations among individuals are at the core of the development process (as at the core of the entire organization) and those technologies might assist in improving them.

In a DevOps approach, development and operation[2] teams should exchange information and work together as much as possible. A team is responsible for the subproduct for its whole lifetime and there should not be a handover from developers and infrastructure operators. In DevOps is also central to focus on the process instead of just on the product; since the spotlight is on the process, automation is a resource to be exploited as much as possible to enhance and simplify it. If there are repetitive tasks, that could be automated, people can work more efficiently.

In DevOps, automation also enables Continuous Integration and Continuous Delivery. *Continuous Integration* (CI) is the process of integrating frequently new code written by developers. This is in contrast to having developers working on independent feature branches for weeks; long periods of time in between a merge and another mean that a lot of code has already changed, the goal is instead to avoid integrations problems that come from large and infrequent merges. When changes are committed and merged the tests automatically start running; failing tests means the build is broken and it needs to be fixed. With this kind of workflow, problems can be identified and quickly fixed. *Continuous Delivery* (CD) *"is the ability to get changes of all types—including new features, configuration changes, bug fixes, and experiments—into production, or into the hands of users, safely and quickly in a sustainable way"* [16]

1.1.1 Influence on MLOps

MLOps derives its main principles from DevOps but some differences exist: both are aimed to reduce the time to get the system into production by simplifying the lifecycle and by standardizing it. This goal is achieved by letting teams collaborate

[2] *Operations* is about the managing of the infrastructure and the services it hosts.

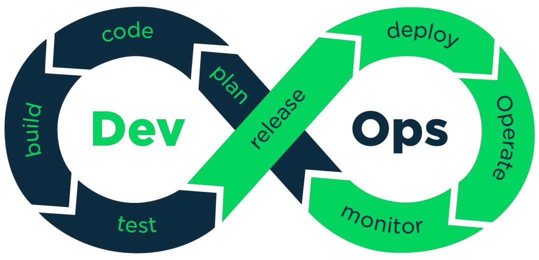


Figure 1.1: DevOps lifecycle

together and by enhancing automation. In DevOps and MLOps a team is responsible for the subproduct for its whole lifetime and there should not be a handover from developers and infrastructure operators. Though in MLOps, teams need to incorporate Machine Learning researchers and data scientists who are often not experienced software engineers. Due to the different nature between DevOps and MLOps also the approach to tests is different: since DevOps requires tests for the code, MLOps requires tests also for data validation, model validation, model quality. Both the approaches do not impose to use some specific tool, but in the two cases choose the right instrument is crucial to reach the goals.

Another important area that deviates MLOps from DevOps is how Continuous Integration/Continuous Delivery (CI/CD) pipelines are constructed. In MLOps, CI components need to extend to testing and validating data schemas, data, and models. CD components need to support the deployment of the training pipeline as well as the final model prediction service or application. Additionally, there is another component, Continuous Training (CT), that needs to be accounted for to enable automatic model retraining and refinement. The process in Figure 1.1 is part of the MLOps workflow as well, but the latter adds some more steps related to data and model management.

1.1.2 MLOps, DataOps, ModelOps and AIOps

In the years, since the popularization of DevOps, a lot of ops-terms born: terms as SecOps (for security), NetOps (for networks), ITOps or GitOps. Referred to data, besides MLOps, emerged terms as DataOps, ModelOps and AIOps (other terms can be less frequently used).

DataOps This term is often used as a synonym of *MLOps* but it is slightly different. *"The main tasks in DataOps include data tagging, data testing, data pipeline orchestration, data versioning and data monitoring. Analytics and Big Data teams are the main operators of DataOps"* [17]; other people who can adopt DataOps

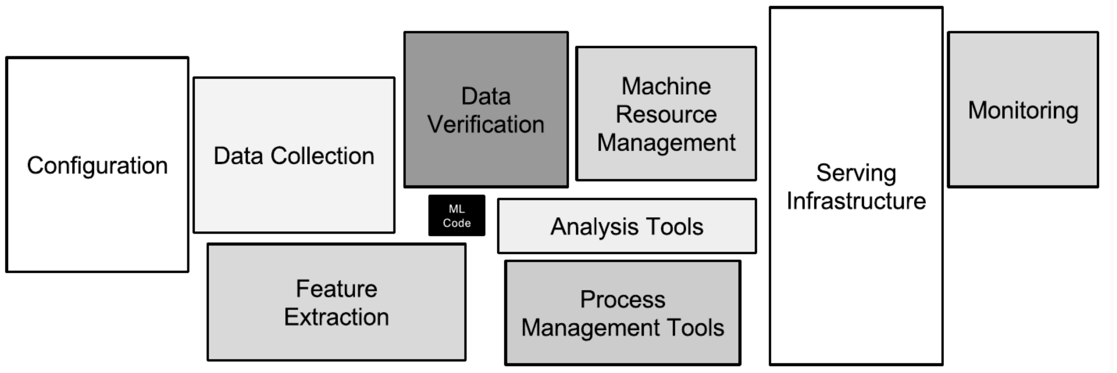


Figure 1.2: Components of a Machine Learning system [1]

could be data analysts, BI analysts, data scientists or data engineers. The DataOps manifesto strictly reminds the Agile Manifesto and it can be resumed as *"Individuals and interactions over processes and tools; working analytics over comprehensive documentation; customer collaboration over contract negotiation; experimentation, iteration, and feedback over extensive upfront design; cross-functional ownership of operations over siloed responsibilities"* [18].

ModelOps Also ModelOps and MLOps are often used interchangeably. ModelOps is more general than MLOps: it's not only about Machine Learning models but about any kind of model.

AIOps AIOps can be confused with MLOps but it refers to the process of solving operational challenges through the use of Artificial Intelligence.

1.2 The Machine Learning lifecycle

A common feature between DevOps and MLOps is that they both highlight the process prior to the product. The Machine Learning lifecycle does not involve only the model building: the infrastructure of a Machine Learning system is vast and complex and only a small fraction of it is composed of the code for the Machine Learning model (the black component in Figure 1.2). For this reason, is crucial to underline that MLOps is focused on the whole Machine Learning lifecycle and not only on the model building phase. In Data Science the system lifecycle can be declined in different forms according to the specific field (the process can spotlight the business understanding, as in CRISP-DM, or it can, for example, favour the data), thus they all share some common steps. Figure 1.3 shows the steps of the Machine Learning lifecycle.

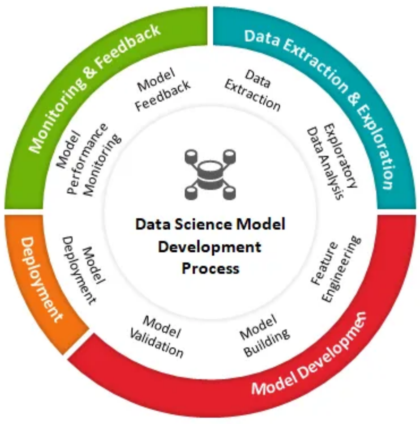


Figure 1.3: Machine Learning lifecycle [2]

Data Extraction and Exploration *Data Extraction* is typically the first step in Data Science projects: the data in various formats need to be extracted and cleaned to be used for further analysis. Once the data are ready they can be explored to understand the hidden patterns. The *Data Exploration* stage can include documenting how the data was collected, looking at summarizing statistics of data, taking a closer look at the distribution of the data, finding correlations and cleaning reshaping, filtering, sampling the data.

Model Development The very first step of the *Model Development* phase is to apply appropriate transformations on the data to enhance them and to make them fit for Machine Learning algorithms: Feature Engineering. Adding more features may produce a more accurate model, but it also comes with downsides: the model can become more expensive to compute, more features require more inputs, more feature means a loss of stability.

In an MLOps approach, it would be useful to automate feature selection, by using heuristics, to estimate how critical some features will be for the performance of the model. This deal also favours the adoption of a Feature Store, a set of repositories of different features associated with business entities that are created and stored in a central location for easy reuse. Once the data are prepared, the model (or many models) can be built by applying Machine Learning algorithms and by feeding them with the data themselves. All the Model Development stage includes assessing how good a model can be built, finding the best hyperparameters, tuning the tradeoff between underfitting and overfitting and also finding a balance between model improvement and computation costs. This step also concerns experimentation, which takes place throughout the entire Model Development process: every important decision comes with at least some experiment. When the models are built they have to be validated to ensure they perform as expected. *"Essentially all models are wrong, but some are useful" - George E.P. Box (20th century British statistician)* [19]. It's important to evaluate a model and compare its performances to what existed before; this can be achieved by using metrics, but there is no one-size-fits-all metric. In Model Development different tasks may be repeated and automation can simplify the process. An MLOps attitude can also provide tools to track hyperparameters, version the different models, log metrics and simplify models comparison.

Deployment In this step, the models are taken from their original development environment and integrated into business applications. There are two main kinds of *Model Deployment*: *Model-as-a-service*, in which the model is deployed into a framework to provide a REST API endpoint (that responds to requests in real-time) or *Embedded model.* This last type is the most simple approach: we treat the model artefact as a dependency that is built and packaged within the consuming application. An MLOps approach can bring to the Deployment phase both CI/CD and containerization.

Monitoring and Feedback Last but not least is the step about *Monitoring and Feedback*: when the model is deployed in production it is crucial that it continues performing well, thus it needs to be observed and audited to avoid (or prevent) any kind of drift. Eventually, the performance will degrade and they will be unacceptable, in these circumstances the model retraining will be necessary. Adopting MLOps can simplify and easy to reproduce the process to rebuild and redeploy a model.

1.2.1 A process, not only a product

Figure 1.3 points out that each step of the process may have, within it, other steps, which could be considered as a "process in the process". 'Data extraction and Exploration' may be formed of two distinct phases, the 'Model Development' step may cover Feature Engineering, Model Building and Model Validation. In the image also the 'Monitoring' phase is made of two steps: Monitoring and Feedback. The image is not exhaustive: the lifecycle could cover other stages according to each single circumstance. This perspective of the lifecycle, seen as a composition of modules (and submodules), allows to pipeline the process and automate it. We may also have different pipelines: for example, we can have a *Training pipeline*, for the Training phase, and a *Prediction pipline* for the 'Model Deployment' and 'Monitoring'. Using a modular approach for the lifecycle allows data scientists also to reuse each component in distinct pipelines (for example we may need to run the Feature Engineering step both in the *Model Development/Training pipeline* and in the *Prediction pipeline*). This point of view of the lifecycle, seen as a set of pipelines made of reusable and modular components, enables three main concepts about automation in MLOps: Continuous Integration, Continuous Deployment and Continuous Training.

1.2.2 People involved in the Machine Learning lifecycle

Even though Machine Learning models are primarily built by data scientists, the entire lifecycle of a Machine Learning system involves many people from different teams. One of the main features of MLOps is to foster collaboration between teams: MLOps can affect everyone working on the Machine Learning lifecycle and, improving collaboration, provides benefits on avoiding the silos between different teams. Various roles can be involved in the lifecycle of a Machine Learning system, the main ones are Subject Matter Experts, Data Scientists, Data Engineers, Software Engineers, DevOps. A new role arises with MLOps: the MLOps Engineer.

Subject Matter Experts This is the first role involved when the Machine Learning lifecycle starts, and the *Subject Matter Experts* must be committed during all the process. Data-oriented profiles tend to lack a deep understanding of the business and the problems that need to be addressed. A Subject Matter Expert defines the goals, the business needs and the Key Performance Indicators (KPIs) that they want to achieve or address. This figure has a role, not only at the beginning of the process but at the end as well. Sometimes, to understand if a Machine Learning is performing well or as expected, traditional metrics (accuracy, precision, recall, etc.) are not enough and data scientists need feedbacks from the Subject Matters Experts. For example, data scientists could build a model that has very high accuracy in a production environment but doesn't provide the expected results from a business point of view. When building an MLOps process, it's critical to provide to the Subject Matter Experts an easy way to understand deployed model performances in business terms. That is not just about metrics but also about the results or the impact of the model on the business process. It also would be useful to provide a way to dig into individual decisions made by a model to understand why it came to that decision (such as model interpretation and explanation).

Data Scientists Often the role of the *Data Scientist* in the Machine Learning lifecycle is read as strictly related to the model building step, actually it is wider: Data Scientists need to be involved with Subject Matter Experts, understanding and helping to shape business problems in such a way that they can build a valuable solution. A Data Scientist does not only need technical skills but he/she also needs to communicate effectively with other people involved in the process, people from he/she are often siloed. A robust MLOps system should help to facilitate and simplify collaboration between Data Scientists and other profiles with suitable organizational infrastructure. Building good MLOps practice should also allow Data Scientists to quickly take action on the deployed models.

Data Engineers The role of the *Data Engineers* in the lifecycle is to optimize the retrieval and to use data to eventually power Machine Learning models; this means working closely with Subject Matters Experts to identify the right data and also prepare them for use. They also work closely with Data Scientists to resolve any data issue that might cause a model to behave undesirably in production.

Software Engineers *Software Engineers* are involved in building classic software and applications and it is important that they work together with Data Scientists to ensure the functioning of the whole system. For example, Machine Learning code has to fit into the CI/CD pipeline that the rest of the software is using (think to a Machine Learning model, built by Data Scientists, which needs to integrate itself with the app or the website used by the very last users).

DevOps MLOps was born out of *DevOps* principles, but they can coexist together. DevOps, within the Machine Learning lifecycle, are people smoothing the transition from development to operations by maintaining the infrastructure. They must ensure security, performance and availability of the Machine Learning models; they also are responsible for bridging the gap between traditional CI/CD and Machine Learning CI/CD. Because of these two roles, DevOps require tight collaboration with Data Scientists as well as Data Engineers.

MLOps Engineers The rise of MLOps introduced a new role: the *MLOps Engineer*. An MLOps Engineer is someone with enough knowledge of Machine Learning models to understand how to deploy them and with enough knowledge of operational systems, to understand how to integrate, scale and monitor models. The MLOps Engineer plays the role of glue between all the other profiles. With one foot on DevOps, the *MLOps Engineer* has responsibilities on the pipeline and in the successful operationalization of the Machine Learning model. With the Data Scientists, the MLOps Engineer tests and deploys models. Dedicated MLOps solutions make the collaboration between all the different profiles more efficient and simple.

1.3 State of the Art and the need for MLOps

MLOps is a really new and young approach and it is constantly evolving; though probably due to its youth, it does not have a clear manifesto and it lacks a certified and a shared definition. A more interesting reason for a lack of a common manifesto comes from the DevOps community and it could be applied also for MLOps: *"it would be the end of the discussion. And that's exactly the problem with a manifesto. [...] The community likes to enhance, educate, enliven, inform and energize people. Basically, everyone is wide open to embracing anything that helps out"* [20].

MLOps is very flexible also about technologies. Today cloud providers, such as Google, Amazon, Microsoft or Databricks, offer solutions for adopting MLOps. There is also copiousness of Open Source tools on the stage and the respective communities are highly active in enhancing their tools. Today plenty of companies are starting using an MLOps-based approach to develop a Machine Learning system, but unfortunately, in Italy, MLOps is still not (or little) known.

Through a GoogleTrends search, it becomes clear that the MLOps topic is climbing in interest from both a scientific and practical perspective. Google Trends puts MLOps as one of the most promisingly increasing trends [21] (Figure 1.4).

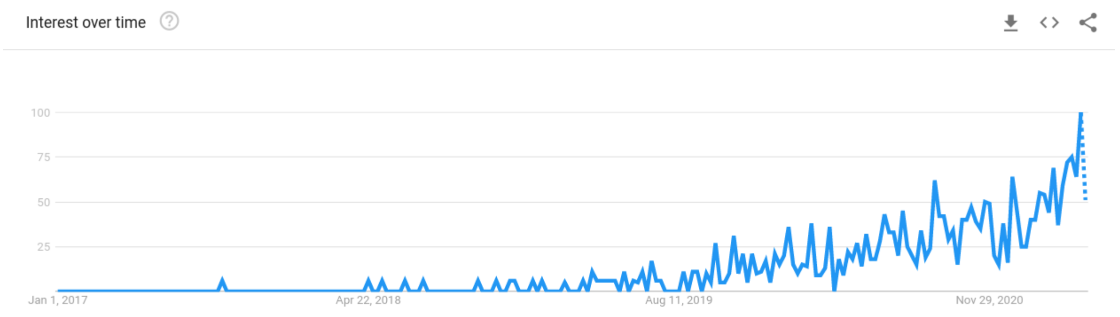


Figure 1.4: Trends for MLOps searches, January 2017 - May 2021

"As detailed in a recent Cognilytica report on MLOps [22], increasingly the market is seeing the emergence of MLOps solutions designed to simplify the usage and consumption of various AI and ML models. These solutions will become increasingly required as the bulk of the market adopts AI [...]. The MLOps market is relatively immature and nascent, with technology solutions emerging only in the last year or two [...]. Indeed, it's been predicted to be a major trend even for 2020". Moreover, according to Cognilytica, the MLOps market is expected to

expand to nearly US$4 billion by 2025 [23]. *"As a result, IDC reports, 28% of AI/-machine learning projects fail, with lack of necessary expertise, production-ready data, and integrated development environments cited as the primary reasons for failure.3 Many more projects (47%) fail to even make it out of the experimental phase and into production"* [24]. In March 2021 the Kubeflow Community conducted a survey [3] on benefits, gaps and requirements for Machine Learning, by also outlining the need for MLOps. According to the survey, the majority of Machine Learning models have a fairly short life: 50% run in production for 3 months or less. On the other end of the spectrum, 25% of the models remain in production for 6 months or longer.

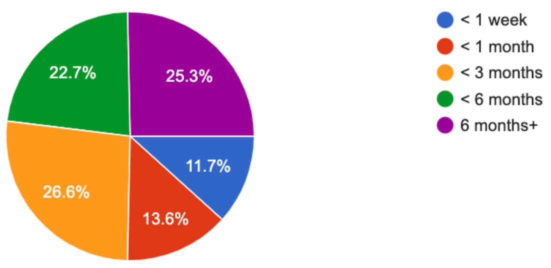


Figure 1.5: Machine Learning models life time [3]

"Beyond ML codes, these frameworks and platforms have provided functional components and utilities to help avoid ongoing maintenance costs brought by hidden Technical Debt in Machine Learning systems" [25] MLOps comes into play to face different "Technical Debts", anti-patterns and challenges common in developing and deploying a Machine Learning system.

1.3.1 Technical Debt in Machine Learning systems and anti-patterns

"Technical Debt" is a term related to immature, incomplete or inadequate code (due to design deficiencies, low quality or other problems), which will require additional work to be fixed. It is a metaphor linked to finance: having technical debts on software would be like paying interest on a loan. In the popular paper *"Hidden Technical Debt in Machine Learning Systems"* are summarized some pitfalls in operating ML-based systems into production. *"Technical debt may be paid down by refactoring code, improving unit tests, deleting dead code, reducing dependencies, tightening APIs and improving documentation. The goal is not to add new functionality, but to enable future improvements, reduce errors and improve maintainability"* [1]. Machine Learning systems have a special capacity for

[3]The survey collected 179 responses

incurring in Technical Debt, because they have all of the maintenance problems of traditional code plus an additional set of ML-specific issues. Some Technical Debt may be related to data dependencies, model complexity, reproducibility, testing, monitoring and dealing with changes in the real world. Kaz Sato summarizes, in his talk at Cloud Next'18 [26], some Technical Debts and anti-pattern, classified as Development, Deployment and Operation anti-patterns.

Development anti-patterns

Super-hero A *Super-hero* is a Machine Learning researcher, a Data Scientist or an engineer (or someone else) who has a skill set on preparing data, on building the model and also on operating the infrastructure by using Kubernetes and Docker; he/she manages all the system lifecycle from its start to the production phase. The Super-hero has essentially two problems: he/she does not scale and the know-how about the system is pinpointed in a single person; for this second reason the knowledge could be lost, for example, when the super-hero changes job or when she/he is staffed in a different team. A solution for this anti-pattern could be found in building a scalable team and split the roles into much simpler ones.

A black-box that nobody understands Sometimes, due to the teams' silos and the handover between people involved in the lifecycle, it is possible that no one understands the whole system. It's important to make everything interpretable by humans and to have not a "black-box". A solution could be an approach where engineers and researchers are embedded together on the same teams. This attitude leads to tight and close communication and lets to share all the results.

CACHE principle In a Machine Learning system, generally, is not possible to make isolated changes. A change to one feature could affect all of the other features, or a change on a hyper-parameter could affect the whole result. More generally, if the model is changed, we have no guarantee of the model will keep generalizing well. This is called *CACHE principle*: Change Anything Changes Everything. *"Starting with an interpretable model makes debugging easier"* and *"Keep ensembles simple"* [27] could be two approaches to deal with the CACHE principle. Some models are easier to understand and to interpret, but a drawback could be a loss of accuracy on the performances of the model itself (for example Deep Learning is attractive and, dealing with complex problems, leads to better results than other approaches; but, as a flaw, a deep model is more difficult to understand). If simple models are not enough for the performance requirements using Ensembled Models could represent a solution for the CACHE principle.

Deployment anti-patterns

Lack of Machine Learning lifecycle management Figure 1.2 shows the elements of a Machine Learning system and it underlines that the whole lifecycle is more complex than dealing with only the model building. In the diagram, the rest of the system is composed of configuration, automation, data collection and verification, serving, monitoring. Due to the silos between different teams a lack of Machine Learning lifecycle management may occur. The lifecycle itself must instead be handled as an end-to-end process and not as a set of independent tasks. The solution for this anti-pattern is to have integrated job management and a job orchestrator, so we can define all the components as modules of a pipeline; dealing with this approach everything is built as continuous deployment and repeatable infrastructure, not depending on a team or a person only.

Lack of data validation In an IT system, the behaviour is defined by the code, in a Machine Learning system (also) by the data; in the first, validation can be achieved by unit tests, though in a Machine Learning system is harder to deal with validation. The solution is to use data validation tools, integrated into the system and to use them as a step (or steps) of the pipeline which manages the Machine Learning lifecycle.

Operation anti-patterns

Lack of continuous monitoring Any kind of model will see a drop in accuracy over time; it may take years or maybe quarters or months or hours, but the accuracy will drop. There must be practices to monitor the model in production and to update it quickly.

Training-serving skew Any single difference in the training data or in the preprocessing method between training and serving can change the accuracy.

Not knowing the freshness requirements Each Machine Learning system has its own freshness requirements. According to the different business domains, we may want to update our model every minute, day, week, month or year and it is necessary to know this "freshness-time". For example, an advertising application may require a refresh every day, an NLP problem every month or a voice recognition system every year. A solution could be, first, having a large knowledge of the business domain and then, when the model is in production, do analytics on it such to know how much the model performances degrade over time.

1.3.2 Challenges

In a Machine Learning system the real challenge is not about building a model, it is instead to build an integrated system and continuously operate on it in production: namely, handle the system as a process instead of as a product. From this point of view, we may want to automate as much as possible the whole lifecycle and we may desire to be able to reproduce all the steps of the pipeline in a simple fashion. Other desiderata when developing and building a Machine Learning system include:

- Increase collaboration between different teams.
- Choose the right tools from a plethora of different frameworks.
- Validate the data, to face the previously mentioned problem about lack of data validation.
- Reproduce the steps to build the model without effort, whenever we want. Also track, in a transparent way, the parameters used to train a model and its metrics.
- Deal with heterogeneous skills among the people involved in the lifecycle: *"the Machine Learning lifecycle involves people from the business, data science and IT teams; none of these groups are using the same tools or even - in many cases - share the same skills"* [19].
- Test the model (remark that unit tests, from traditional software development, are not enough).
- Cope with *"many dependencies: not only is data constantly changing, but business needs shift as well"* [19].
- Face the Drift problem, *"Change is the only constant in life"* said Heraclitus, the Greek philosopher. The models fail to adapt to changes in the dynamics of the environment, or to changes in the data that describes the environment. A complex challenge is to avoid Data Drift and Concept Drift as much as possible.
- Automatically moves the model from staging to production.

1.4 MLOps features

Before delving into the main concepts in MLOps a secondary one, but no less valuable, needs to be explained: containerization.

Containerization This is a technique used to encapsulate or package up software code (and all its dependencies) so that it can run uniformly, consistently and independently on any infrastructure. Containerization, a lightweight alternative to virtual machines, consists of bundling the application code in a self-contained environment, also with the related configuration files, libraries, and dependencies required for it to run. The single package of software (or container) is abstracted away from the host operating system and it is able to run across any platform or cloud. Containerization is a requirement to enable many of the concepts described below.

1.4.1 Continuous

As underlined before, the Machine Learning lifecycle is not done when the model is put into production. Models need to be monitored and retrained, changes are commonplace and Data and Model Drift may occur. Being *continuous* can be declined in different ways for a Machine Learning system: Continuous Integration, Delivery, Training and Monitoring. An essential ingredient to enable all of these concepts is pipelining the whole lifecycle making use of containers.

Continuous Integration In traditional software *Continuous Integration* is a practice about testing and validating code and components; it also refers to run unit tests when a source code gets changed. The goal of CI is to quickly make sure a new change from a developer is “good” and suitable for further use. Additionally, in MLOps, CI is related to testing and validating data and models. We want each push to the code repository (that contains our training code), to trigger a rebuild of the assets that constitute our Machine Learning pipeline: training containers, hyper-parameters tuning, retrain the model and others. Since in an MLOps-approach the lifecycle is organized in one pipeline (or more), Continuous Integration can be achieved by running the data pipeline and the training pipeline when a change in the code occurs.

Continuous Delivery “*Continuous Delivery* is the ability to get changes of all types — including new features, configuration changes, bug fixes, and experiments — into production safely and quickly in a sustainable way” [28]. In Machine Learning, Continuous Delivery is about delivering an application, based on code, data and models, in small and safe increments, that can be reproduced and reliably released at any time.

Continuous Training It is a new feature, not included in DevOps, concerned with automatically retrain and serving the models. When the data used to train the model(s) change (an update occurs or new data are added) we want to retrain the model itself by automatically triggering the whole training pipeline. *Continuous Training* is a powerful approach to deal with Data Drift.

Continuous Monitoring While the model is in production it may change due to Data Drift or Concept Drift: monitoring concerns with auditing production data and model performance metrics, strictly related to business metrics. We want to understand how the model performs in production and trigger alerts when something goes out of the ordinary and, in this latter case, to rebuild our pipeline and retrain the model.

1.4.2 Reproducibility

Reproducibility is a crucial feature for MLOps (indeed it appears also in the definition of the term). Being reproducible, for a system, enables and favours productivity. Reaching reproducibility is a hard challenge: in traditional software the code is static and it is straightforward to reproduce it; in Machine Learning the data and the models eventually will change and different mechanisms are needed to repeat actions executed in the past. Reproducibility also paves the way to automation. When we are able to reproduce the steps and the whole pipeline of the system, we can automate and enable all the Continuous-related concepts previously exposed.

1.4.3 Versioning and Experiment Tracking

The data will eventually change and a model may be retrained as a consequence of a data update or due to drift (or because a better model is developed). Models may require a revision or their performances may degrade over time. In an MLOps approach is quintessential to keep versioned the data and the models as well. In a Machine Learning system, versioning is harder than in traditional code. The data cannot be versioned using a classical Version Control System (e.g. git); due to the large amount of data involved it requires different techniques: data snapshots may be too large and increment data versioning technologies are in their infancy. Also versioning the model needs a different attitude than code, as the version of a model must track how the model itself is built: used parameters, the environment in which the model is developed, related artefacts and the metrics obtained after the training. In Machine Learning, to obtain good performances for a model, many experiments may be needed. ml-ops.org suggests, as approach, to *"use different (git-) branches, each dedicated to the separate experiment"* [29]. I do not agree with this approach as the branch concepts differ to a specific version (a branch typically includes many commits and versions) and it would lead to an explosion of the number of branches in the repository. Versioning and tracking the experiments, performed to build the models, are also ingredients to enable reproducibility. Using these practices is possible to simply retrain a model in the same way of previous training and using the same data (and, as consequence, it is easy to compare different models with each other).

1.4.4 Testing

In Machine Learning testing the system does not involve only the code, it includes also testing the features and data, the model development and the infrastructure. While some aspects are inherently non-deterministic and hard to automate, different kinds of automated tests can add value and improve the overall quality of the system. Testing includes:

- Validating data and features: the input data must be validated against the expected schema, with assumptions about their valid values.

  Feature creation code should be tested by unit tests (to capture bugs in features) and the data should be policy-compliant (e.g. GDPR). These requirements should be programmatically checked in both development and production environments. Further, features importance tests may be useful to understand whether new features add predictive power.

- Validating the model quality: test for the Machine Learning training should verify that algorithms make decisions aligned to the business objective: namely algorithm loss metrics (MSE, loss, etc.) should correlate with business impact metrics.

- Validating model bias and fairness: the performances of a model and testing the fairness/bias/inclusion of the training data is needed; for example, there might be unbalanced data for a given feature (e.g. gender or region) compared to the actual distribution.

- Avoid stale models: the system, in production, must not have stale models, which may affect the quality of the prediction.

- Value the trade-off between performances and the model complexity: it is necessary assessing the cost of more sophisticated models (e.g. linear model vs neural network).

- Test the infrastructure: the training of the ML models should be reproducible (that means that training the model on the same data should produce identical results). The architecture must be stress-tested and the full Machine Learning pipeline should be integration-tested. Before serving the model it must be validated; the model in the training environment must give around the same score as the model in the serving environment.

1.4.5 Monitoring

In a Machine Learning system, the performances of a model degrade over time, the data change and the model itself needs to be retrained. Once the model has been deployed it requires to be audited to assure that it performs as expected in production. Monitoring involves checking data invariants and set up alerts to notify if input data does not match the schema; it concerns controlling the numerical stability of the model as well (and trigger alerts for the occurrence of any NaNs or infinities). One of the previously mentioned anti-patterns was about training-serving data skew: what data is being fed to the models is the object of monitoring. Another field for monitoring is the degradation of the predictive quality of the model on served data. Also conduct auditing on the user actions is primary: based on further user actions, reward metrics may be captured to understand if the model is having the desired behaviour. For example, if the system shows product recommendations, it can track when the user decides to purchase the suggested product as a reward. Two other objects of monitoring are how stale the system in production is (by measuring the age of the model: older is the model itself more it will tend to decay in performance) and the model fairness. This last concept refers to analyzing input data and output predictions against features that could bias, such as race, gender or age.

1.4.6 Modularity

In MLOps, from the point of view of the architecture of the system, the components need to be loosely coupled. This key architectural property enables teams to easily test and deploy individual components. Having a loosely coupled architecture allows different teams to work independently, without relying on other teams. An essential ingredient to achieve modularity is containerization, which paves the way to handling the workflow as a pipeline composed of different modules.

1.4.7 Automation

Automation is quintessential in MLOps and its level defines the maturity of the Machine Learning process, which reflects the velocity of training new models or training new ones. Google identifies three levels of automation [4] for a Machine Learning system: in the first (level 0) all the process is handled manually, level 1 includes the execution of the model training automatically and, in the final stage, a CI/CD system is introduced. The three steps do not have to be immediately and simultaneously implemented, these practices can be gradually realized to help improve automation of the Machine Learning development and production system.

Level 0: manual process Level 0 is referred to as a process, for building and deploying Machine Learning models, entirely manually (namely without MLOps). This is the most basic level of maturity and it is common in many businesses that are beginning to apply Machine Learning to their use cases. This manual approach might be sufficient when models are rarely changed or trained; in practice, models often break when they are deployed in the real world. By using this approach the execution of each step is manual: including data analysis, data preparation, model training and validation and the transition from one step to another as well. Machine Learning and operations are totally disconnected and the data scientists hand over a trained model, as an artefact, to the engineering team to deploy it. Release iterations are infrequent and due to the long time to get the model in production problems related to training-serving skew may occur. Continuous Integration and Continuous Delivery are not adopted because few changes are assumed. This approach also might lead to a lack of active performance monitoring.

Level 1: Machine Learning pipeline automation The main goal of level 1 is to perform Continuous Training of the model by automating the Machine Learning pipeline. This approach enables rapid experiments, thanks to the orchestrations of the steps to train and build the model. The transition between different steps is automated, this allows rapid iteration of experiments. By embracing Continuous Training the model is automatically trained in production, using fresh data based on live pipeline triggers (incoming new data can trigger a new run of the training pipeline or scheduled triggers can be used as well). Another central feature of this level of automation is the symmetry among experimental and operational environments: the pipeline implementation, used in the development or experiment environment is used also in the preproduction and production environment. This is a key aspect of MLOps practice. To construct Machine Learning pipelines, components need to be reusable, composable and, potentially, shareable across the pipeline (or different pipelines). Modularization becomes quintessential in this level of automation: it allows to decouple the execution environment from the custom code runtime and to isolate each component in the pipeline (components can have their own version of the runtime environment and can be implemented through different languages and libraries). In level 0 a trained model is deployed as a prediction service, in level 1 a whole training pipeline is deployed. Additional components may be used in this level of automation:

- Data and model validation.

- Feature Store: it is a centralized repository where we standardize the definition, storage and access of features for training and serving. It helps discover and reuse available features instead of recreating them.
- Metadata management: information about each execution of the pipeline should be recorded in order to enhance reproducibility and comparisons. Each time the pipeline is run different metadata may be recorded; for example, can be logged the pipeline and component version that is executed, the start and end time (and the execution duration as well), the parameters passed to the pipeline or also the metrics produced during the model evaluation step.
- Pipeline triggers: the pipeline can be run according to different events: manually on demand, on a schedule (each hour, each day, each month), on availability of new training data, on performance degradation or also on significant changes in the data distribution.

Level 2: CI/CD pipeline automation The final stage involves the full automation of the CI/CD system which implements the Machine Learning pipeline. Setting up a CI/CD system enables to automatically test and deploy new pipeline implementations. This approach allows coping with rapid changes in the data and business environment. The whole pipeline consists of the numbered stages in Figure 1.6:

1. This phase concerns the development and the experimentation. New Machine Learning algorithms are explored and new models are built. The output of this stage is the source code of the pipeline steps, ultimately pushed to a source repository.
2. Then Continuous Integration of the pipeline comes into play. The source code is built as various tests as well. The outputs here are pipeline components to be deployed later.
3. After Continuous Integration the artefacts produced are deployed to the target environment.
4. With all these steps implemented the pipeline can be automatically executed in production according to a schedule over time or in response to a trigger. The output of this stage is a trained model that is pushed to the model registry.
5. The trained model is served as a prediction service and exposed for the predictions.

6. The last step involves monitoring: statistics on the model performances, based on live data, are collected. Triggers to execute the pipeline or a new experiment cycle will be executed as a consequence of changes in the performances.

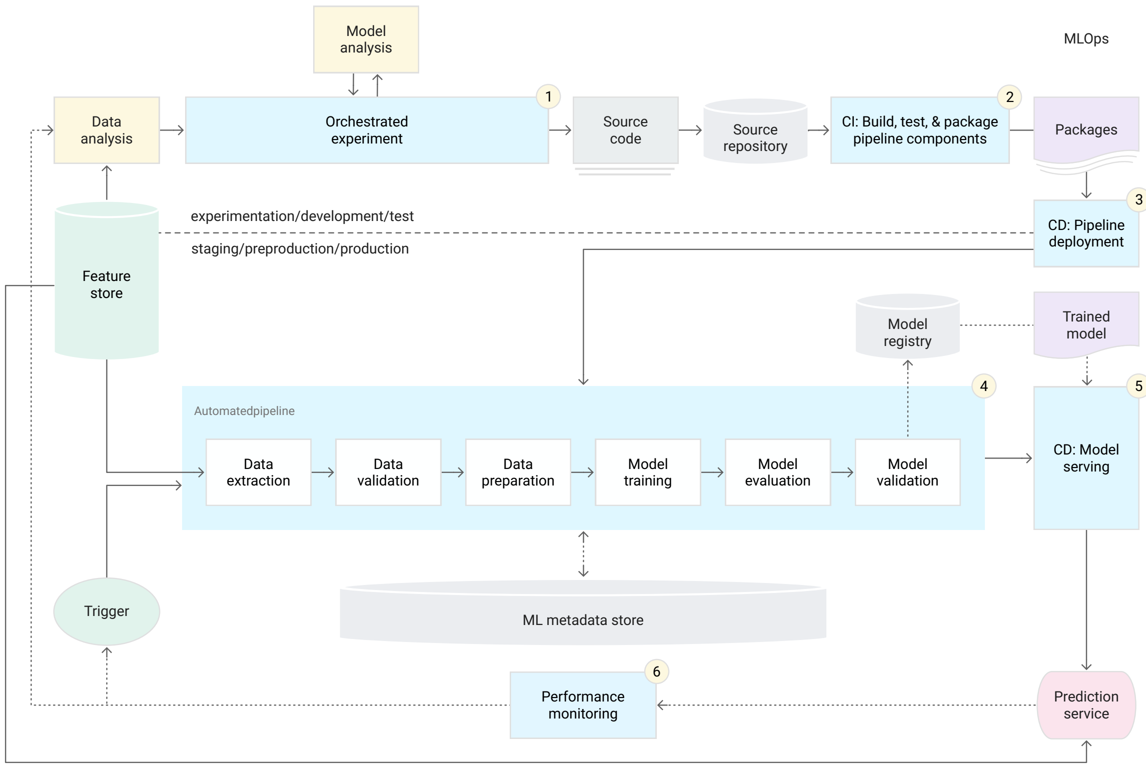


Figure 1.6: Level 2: CI/CD and automated ML pipeline [4]

1.4.8 Workflow Pipeline Design Pattern

A Design Pattern is a solution for a problem that occurs over and over again, in such a way the solution itself can be adopted many times. Machine Learning Design Patterns [30] book explains MLOps as a Design Pattern called "*Workflow Pipeline*". The approach covers all the features of MLOps explained above, by especially emphasizing and focusing on creating an end-to-end reproducible pipeline by containerizing and orchestrating the steps in our Machine Learning process. The Workflow Pipeline Design Pattern highlights that monolithic apps should be replaced in favour of a microservices architecture, where individual pieces of business logic are built and deployed as isolated (micro-)packages of code. With microservices, a large application is split into smaller, more manageable modules so that developers can build, debug, and deploy pieces of an application independently. Without running our Machine Learning code as a pipeline, it would be difficult for others to reliably reproduce our work. The Workflow Pipeline pattern lets others run and monitor our entire end-to-end process in both on-premises and cloud environments. Containerizing each step of the pipeline ensures that others will be able to reproduce both the environment we used to build it and the entire workflow captured in the pipeline (potentially also months later). This approach allows faster development and minimizes the risks associated with a monolithic process. The Workflow Pipeline pattern comes with a Directed Acyclic Graph (DAG), for this reason we can enjoy a flexible workflow environment: we have the option of executing individual steps or running an entire pipeline end-to-end. This also gives us logging and monitoring for each step of the pipeline across different runs and, additionally, enables tracking artefacts from each step of the workflow.

Chapter 2

A survey of technologies for MLOps

MLOps does not enforce an implementation through a specific set of tools, though choosing the right instrument (according to business requirements) may be crucial to building a good MLOps infrastructure. A large number of tools are available to reach different goals and to deal with various challenges explained in Chapter 1. In the following will be discussed some of the most used Open Source tools; different solutions by the main cloud vendors (especially Google [4] [31], Amazon [32], Microsoft [33] and others) have been developed as well, but they will not be explored here. In most cases, Open Source tools offer solutions to deal with specific problems (experiment tracking, CI/CD, pipeline orchestration, etc.), while cloud vendors' tools provide full environments to deal with MLOps. Remarkably, commercial solutions often integrate the Open Source tools; for instance, the solutions offered by Databricks and Google respectively integrate the Open Source tools MLFlow and Kubeflow.

2.1 Open Source technologies

Using Open Source software in a business environment always requires to evaluate a trade-off: on one hand the reliability of the tool (and its outlook), for example a new project can be abandoned after a while because the tool does not grow as expected; on the other hand, it may be subject to the work of a large community and, consequently, it may be continuously improved over time. A strong community is one of the most valuable elements for an Open Source software, both for its growth and for its solid users' support. Unfortunately, a common drawback for the Open Source software mentioned below is a lack of quality in the documentation.

2.1.1 Environment/Containerization

Docker Docker is an Open Source platform for developing and running applications, leveraging on the idea of containerization: it provides the ability to package and run an application in a loosely isolated environment, i.e., a container. Docker enables the separation of the applications from the infrastructure and also allows to significantly reduce the delay between writing code and running it in production. Docker containers can run on a developer's local laptop or on a cloud environment. Docker portability and lightweight nature also make it easy to dynamically manage workloads, scaling up applications and services (as business needs dictate), in near real-time.

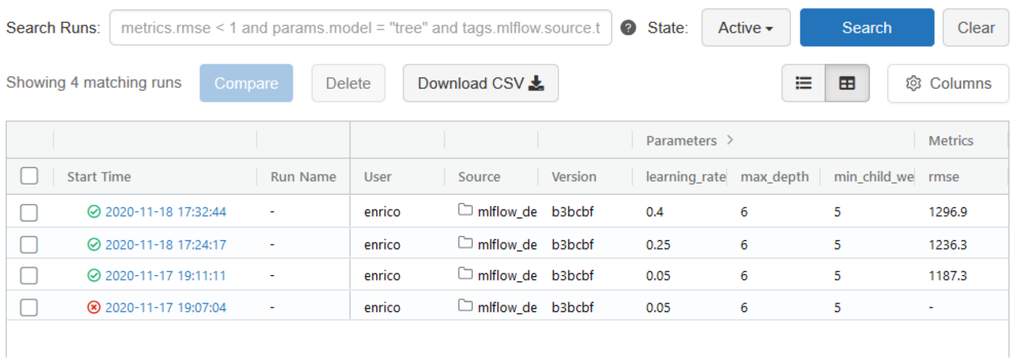
In an MLOps infrastructure, Docker allows packaging all the steps of the pipeline in microservices. This approach enables to reproduce the pipeline stages (or a single container) in different environments and, also, along with Kubernetes and other tools, to build a CI/CD Machine Learning pipeline.

Kubernetes Kubernetes is an Open Source orchestration framework, which helps to manage applications made of a large number of (docker) containers in different environments (e.g. physical machines, virtual machines, cloud, hybrid environments, etc). Kubernetes allows to maintain consistency across development, testing, and production phases; it also enables to embrace a microservice-based approach (instead of building monolith applications). As a consequence, it allows handling loosely coupled, distributed and modular systems, providing high availability and scalable architecture. "*The Design Pattern of Kubernetes is that infrastructure definitions are declarative and new versions of a resource definition force a reconciliation process to change the infrastructure running on the cluster to eventually reflect the current definition. The process allows for the beneficial "GitOps" pattern to be followed where every version of a resource is committed to source control (e.g., Github)*" [34].

Kubernetes paves the way for a distributed environment and, in MLOps, is widely used (also along with Docker) as a low-level tool to enable orchestration and Machine Learning CI/CD. On top of Kubernetes different tools are used, such as Kubeflow (for the pipeline orchestration) or Jenkins, JenkinsX (for CI/CD) or Seldon Core and KFServing (for serving the model into production).

2.1.2 Experiments tracking

Data science and Machine Learning are iterative processes that require a large number of attempts to reach a certain level of a metric.



Search Runs: metrics.rmse < 1 and params.model = "tree" and tags.mlflow.source.t State: Active Search Clear

Showing 4 matching runs Compare Delete Download CSV Columns

						Parameters			Metrics
	Start Time	Run Name	User	Source	Version	learning_rate	max_depth	min_child_we	rmse
☐	✓ 2020-11-18 17:32:44	-	enrico	mlflow_de	b3bcbf	0.4	6	5	1296.9
☐	✓ 2020-11-18 17:24:17	-	enrico	mlflow_de	b3bcbf	0.25	6	5	1236.3
☐	✓ 2020-11-17 19:11:11	-	enrico	mlflow_de	b3bcbf	0.05	6	5	1187.3
☐	✗ 2020-11-17 19:07:04	-	enrico	mlflow_de	b3bcbf	0.05	6	5	-

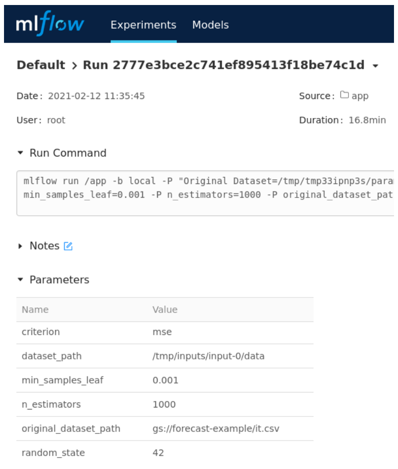
Figure 2.1: Example of different runs of a model training and the tracked parameters and metrics

MLFlow MLFlow is an Open Source API "*that allows integrating MLOps principles into a Machine Learning project with minimal changes made to existing code. With just a couple of lines of code, you can track all of the details relevant to the project. Furthermore, you can even save the model for future use in deployment, for example, and you can compare all of the metrics between individual models to help you select the best model*" [35]. In Machine Learning, tracking an experiment requires to record the environment where the experiment itself takes place. Another remarkable piece of information, which has to be tracked, regards all of the packages and the dependencies used to build a model. This tool provides a way to package the Machine Learning code in a reusable and reproducible fashion; this deal allows to share the code (and its environment) with other data scientists (and enhance collaboration). Another characteristic of MLFlow is to provide a central Model Store to collaboratively manage the models. Below are listed the main features of MLFlow:

- MLFlow Tracking: this element of MLFlow enables reproducibility, automation, allows to make experiments comparable and filter experiments according to different criteria. Figure 2.2 and Figure 2.1 show two examples of the parameters, metrics and other information logged by MLFlow.

- MLFlow Project: this feature allows to track the environment and the dependencies needed to run an experiment. The approach, here, is very similar to a gradle file: in a file, called MLProject, is recorded the *conda* environment and the run entry point.

- MLFlow Models, a standard format for packaging Machine Learning models that can be used in two different ways: real-time serving, through a REST



Default › Run 2777e3bce2c741ef895413f18be74c1d

Date: 2021-02-12 11:35:45

Source: app

User: root

Duration: 16.8min

Run Command

```
mlflow run /app -b local -P "Original Dataset=/tmp/tmp33ipnp3s/para
min_samples_leaf=0.001 -P n_estimators=1000 -P original_dataset_pat
```

Notes

Parameters

Name	Value
criterion	mse
dataset_path	/tmp/inputs/input-0/data
min_samples_leaf	0.001
n_estimators	1000
original_dataset_path	gs://forecast-example/it.csv
random_state	42

Figure 2.2: Example of the parameters recorded in a single run of the model training (with also the the experiment duration, the date and the user who executed it).

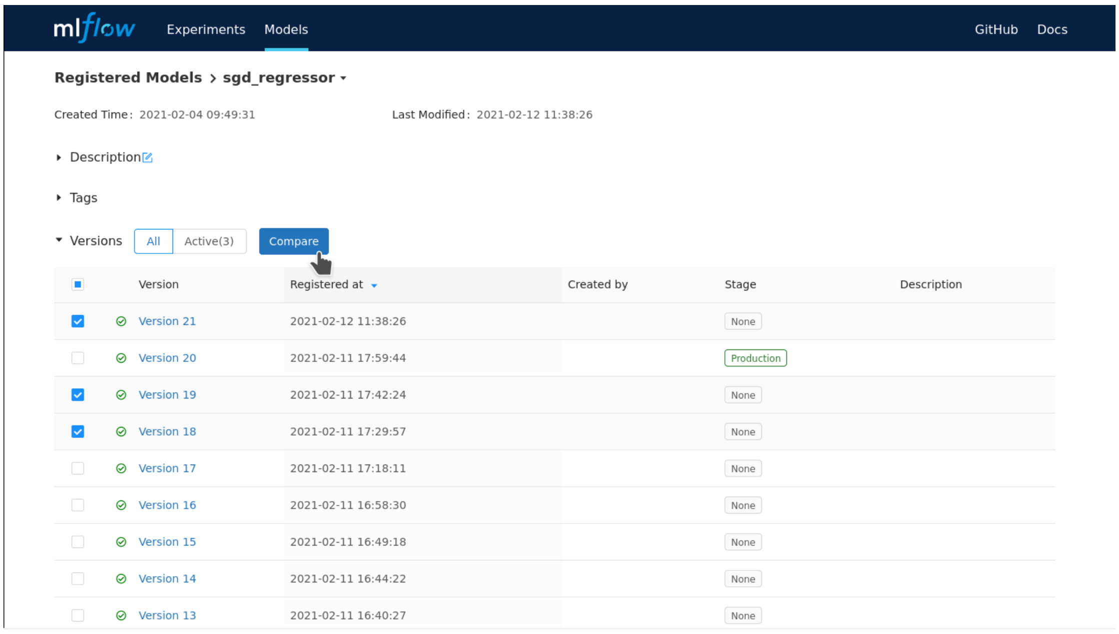
API, or batch inference. Machine Learning models can be saved in different "flavors" [1].

- MLFlow Model Registry: the developed model could evolve over time and needs to be retrained a few times, for example due to a change in the data. Different versions of the model are needed to be managed. MLFlow Model Registry is one of the most powerful features for MLFlow: this component is a centralized Model Store to collaboratively manage and version the trained models. Along with a model, to store and version it, other information are needed; a model version must record the environment in which it has been trained (and its dependencies), the artefacts it produced and the model itself. A registered model has a unique name, contains versions, associated transitional stages and other metadata or additional annotations. Each distinct model version can be assigned one stage at any given time. A model can be "candidate" to be used in production and a model stage can be changed in a programmatic fashion using the MLFlow library (default values are "Staging", "Production" and "Archived").

- Serving: The operation of getting a model into production is always complex and frequently the environment in which the model was built is different from

[1]Flavors are a convention that deployment tools can use to understand the model; this makes it possible to write tools that work with models from any ML library without having to integrate each tool with each library. MLFlow defines several standard flavors, for example, for models developed through Scikit-learn, Tensorflow, Keras, XGBoost, Spark (or other libraries).

the production environment. MLFlow also includes tools for running models locally and exporting them to Docker containers or commercial serving platforms (such as Azure ML or Amazon SageMaker).



Registered Models › sgd_regressor

Created Time: 2021-02-04 09:49:31 Last Modified: 2021-02-12 11:38:26

Description

Tags

Versions All Active(3) Compare

■	Version	Registered at	Created by	Stage	Description
✓	Version 21	2021-02-12 11:38:26		None	
	Version 20	2021-02-11 17:59:44		Production	
✓	Version 19	2021-02-11 17:42:24		None	
✓	Version 18	2021-02-11 17:29:57		None	
	Version 17	2021-02-11 17:18:11		None	
	Version 16	2021-02-11 16:58:30		None	
	Version 15	2021-02-11 16:49:18		None	
	Version 14	2021-02-11 16:44:22		None	
	Version 13	2021-02-11 16:40:27		None	

Figure 2.3: Example of different versions of a model trained through MLFlow. The models can also be compared with each other.

MLFlow provides an API for different languages: Python, R, Java; though it is a very recent tool and unfortunately the API supports all the mentioned features only for Python (only MLFlow Tracking can be used with R and Java).

DVC DVC is a data and Machine Learning experiment management tool whose main characteristic is to be "git-oriented": its approach is to handle parameters, metrics, artefacts, models tracking, and data as well, in the same way code is captured. DVC brings agility, reproducibility, and collaboration into a data science workflow.

DVC consists of a set of commands (some of which very are similar to git commands) that allow tracking all the Machine Learning workflow components (e.g. experiments, parameters, metrics, models, artefacts, log model versions and also data). DVC is built to track everything in a reproducible and easily accessible way. In DVC metrics are first-class citizens and the tool includes a command to list all metric values, to track the progress of an experiment or to pick the best version. DVC also allows to build pipelines of the Machine Learning workflow (for CI/CD) and it provides Data Versioning features as well.

2.1.3 Pipeline Orchestration

Kubeflow Kubeflow project, developed by Google, is dedicated to making deployments of Machine Learning workflows on Kubernetes simple, portable and scalable. This tool is built on top of Kubernetes, so anywhere Kubernetes itself is running Kubeflow is able to be run. Kubeflow was developed to use Kubernetes to standardize and streamline the DevOps work around Machine Learning. The goal of this tool is to make scaling Machine Learning models and deploying them to production as simple as possible, exploiting Kubernetes's potentialities (making it easy, flexible, repeatable, portable deployments, potentially on different infrastructures; deploying and managing loosely-coupled microservices and scaling based on demand). Kubeflow is made of many components, as shown in Figure 2.4, such as Notebooks for spawning and managing Jupyter notebooks, KFServing (and also Seldon and TFServing, which it is integrated to) for deploying Machine Learning models on Kubernetes, Katib for hyperparameters tuning and many others. The main (and the most used) component in this tool is "Kubeflow Pipelines".

Kubeflow Pipelines allow to model the Machine Learning workflow as a sequence of steps, each one can receive data as input and produce one or more outputs. A DAG (Direct Acyclic Graph) defines the ordered sequence of steps and the dependencies among the different components of the pipeline. Each task in the DAG can be visualized through the Kubeflow UI. Kubeflow tracks all the experiments, all their single run and the parameters they used as well (either they have succeeded or not). Along each run different information are logged, such as metrics or the run logs.

A Kubeflow pipeline can be defined by using two different approaches: a simple notebook (by using Kale, another component of the Kubeflow project) or Docker containers. Kale (Kubeflow Automated pipeLines Engine) is a project that aims at simplifying the Data Science experience of deploying Kubeflow Pipelines workflows. Developing and maintaining Kubeflow workflows can be hard for data scientists, who may not be experts in working orchestration platforms and related SDKs. Kale bridges this gap by providing a simple UI to define Kubeflow Pipelines workflows without the need to change a single line of code. Kale allows to "tag" one or more cells of a Jupyter Notebook. Each piece of code with the same tag will be considered a single step of a Kubeflow pipeline. Dependencies among tags (namely Kubeflow pipeline's components) can be also defined. Kale is a very simple approach to define a Kubeflow pipeline but, as such, it does not provide flexibility, composability, reusability and other advantages given by using Docker containers. By adopting this last method each Machine Learning task is conceptualized as a Docker container. Every single container can be aimed at handling, for example, data ingestion, data preparation, model training or model evaluation. Kubeflow, by making use of Docker containers, provides portability, repeatabil-

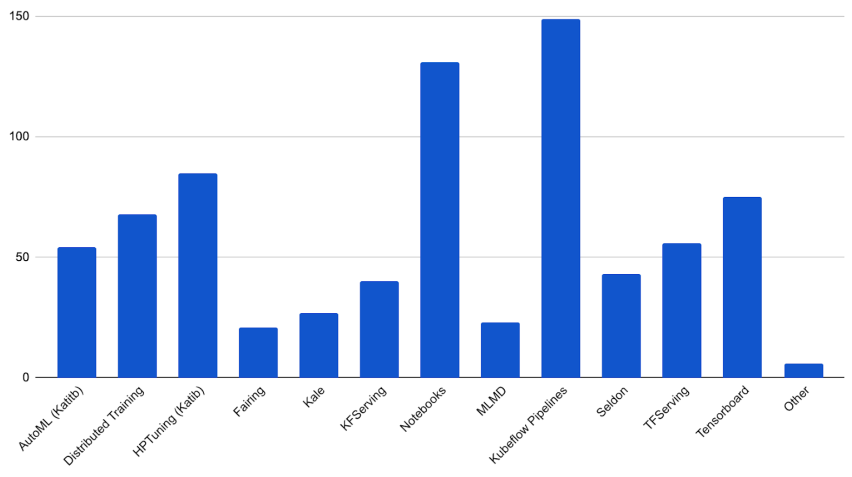


Figure 2.4: Kubeflow components usage (emerged from the survey conducted by the Kubeflow community in march 2021) [3]

ity, reusability, encapsulation, very large flexibility and a modular approach. A container, developed to be run in a pipeline, can be used in more pipelines (for example it would be useful to reuse the same component, to prepare the data, both in the training pipeline and the batch prediction pipeline). By using Docker containers, as an approach, the Kubeflow pipeline can be defined through the Python SDK and its Domain Specific Language. A Kubeflow pipeline is a very powerful tool, which enables, in addition to composability, reusability, flexibility and other mentioned advantages, also Continuous Integration, Continuous Delivery and Continuous Training (if used along with CI/CD tools), key features of MLOps.

Airflow Airflow, by Airbnb, is a general-purpose platform for describing, executing, and monitoring workflows. Airflow pipelines are defined, as code, in Python to get more maintainable, versionable, testable, and collaborative workflows. A drawback in Airflow is that tasks do not move data from one to the other, they only exchange metadata. Airflow workflows' are represented in the form of DAGs and each DAG may or may not have a schedule (cron expressions). Airflow can be integrated into a large number of platforms, such as AWS, Google Cloud Platform, Hadoop, Kubernetes and lots of other tools.

TensorFlow Extended TensorFlow Extended (TFX) is a Google-production-scale Machine Learning platform based on TensorFlow. It provides a configuration framework and shared libraries to integrate common components needed to define, launch, and monitor the Machine Learning system. TensorFlow Extended allows defining pipelines, sequences of components which is specifically designed for scalable, high-performance Machine Learning tasks. Orchestrators such as Airflow and Kubeflow can be used, along with TensorFlow Extended to make configuring, operating, monitoring, and maintaining a Machine Learning pipeline easier.

DVC DVC, exposed in the sections above, among its features, allows defining a pipeline made of different stages. Each stage is associated with a command (to run the step's python script), its dependencies, its parameters and its output. According to the DVC approach, the pipeline is defined in a yaml file, and handled "as code". A DVC pipeline can be also expressed in the form of a Directed Acyclic Graph. DVC pipelines solve a few important problems: automation and reproducibility (the file which defines the pipeline describes what data to use and which commands will generate the pipeline results).

Other tools Elyra AI Toolkit is an Open Source project by IBM, that extends JupyterLab, which has become a standard tool for model development. Elyra provides a visual editor for building Notebook-based AI pipelines, simplifying the conversion of multiple notebooks into batch jobs or workflows. Elyra can also be integrated along Kubeflow.

There are many other tools in the stage, two of them are Kedro, by QuantumBlack (part of McKinsey Company), and Argo, a container native workflow engine for orchestrating parallel jobs on Kubernetes. Kubeflow uses Argo under the hood to orchestrate Kubernetes resources.

2.1.4 CI/CD

Jenkins and JenkinsX Jenkins is an automation server that can be used to handle all sorts of tasks related to building, testing, and delivering or deploying any kind of software. Jenkins can automatically detect code commit in a repository and, consequently, trigger commands (e.g. building a Docker image from a Dockerfile, running unit tests, push an image to a container registry or deploy it to the production server) without manually doing anything. Jenkins is widely used in DevOps and, unlike GitHub Actions, Google Cloud Build or other tools, it is platform-agnostic and self-contained. Jenkins can be installed in many environments such as Linux, Windows, MacOs, Docker and also, as typically happens, in Kubernetes. Jenkins allows defining pipelines, collection of jobs following a particular order or sequence, which can be triggered when an event occurs (for example a push in the code repository). Jenkins can be linked with GitHub, GitLab or Bitbucket repositories.

Jenkins has served as a CI/CD tool for a long time before the emergence of Kubernetes and distributed systems running on cloud native platforms, thus working with Jenkins can be extremely difficult. Recently, with the shift to cloud native and specifically along with the spread of Kubernetes, Jenkins X has emerged as a way to improve, automate, accelerate and simplify Continuous Integration and Continuous Delivery pipelines in cloud environments, so developers can focus on building software. Jenkins can be integrated with other Open Source software such as Grafana (for centralized logs and observability), Tekton (for cloud native pipeline orchestration), Jenkins itself and other tools.

Both Jenkins and JenkinsX can, thus, be used to enable CI/CD and automation: in MLOps the two tools can be adopted to automatically build and run the Machine Learning pipeline.

Other tools Very often also Kubeflow is referred to as a CI/CD tool. In my opinion this assertion is partially correct: Kubeflow enables pipeline orchestration but, at the current state, it does not provide automation (unless using scheduled runs) and a pipeline cannot be triggered when code changes (a primary feature of CI/CD), so it needs to be integrated with other tools to get it.

Also DVC allows to define a pipeline of the Machine Learning workflow and easily reproduce it, but it does not provide automation as well. Another tool, by DVC community, is CML (Continuous Machine Learning); CML is an open-source library for implementing CI/CD in Machine Learning projects by using just GitHub (GithHub Actions) or GitLab (GitLab CI/CD) and a cloud service such as AWS, Azure or Google Cloud Platform.

2.1.5 Feature Store

A Feature is a measurable property of phenomena under observation and (part of) input to a Machine Learning Model. A Feature Store is a new layer of abstraction aimed to reduce the time that Data Scientists spend on getting the data into a format they can use to train models and maximize the amount of time they actually do data science. It is a repository of different features, associated with business entities, that are created and stored in a central location. A Feature Store enables to reuse and share the same features among different business units in a company.

Hopsworks Hopsworks is a complete end-to-end platform for the development and operation of Machine Learning applications, by Logical Clocks, and its main feature is its Feature Store; it is the most popular Open Source software which acts as Feature Store. In Hopsworks, which architecture is shown in Figure 2.5, Feature Store simplifies the end-to-end pipeline, providing an API for data engineers to produce features and an API with which data scientists can easily select features when designing new models.

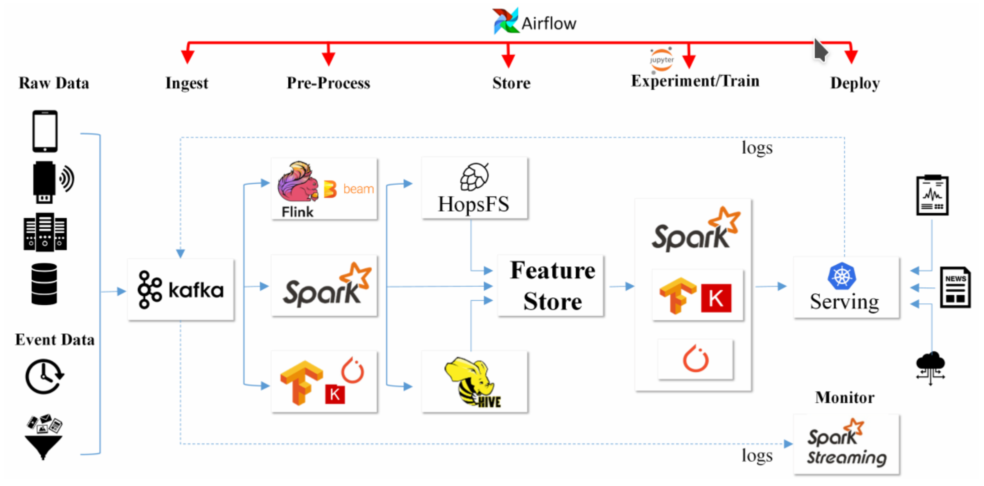
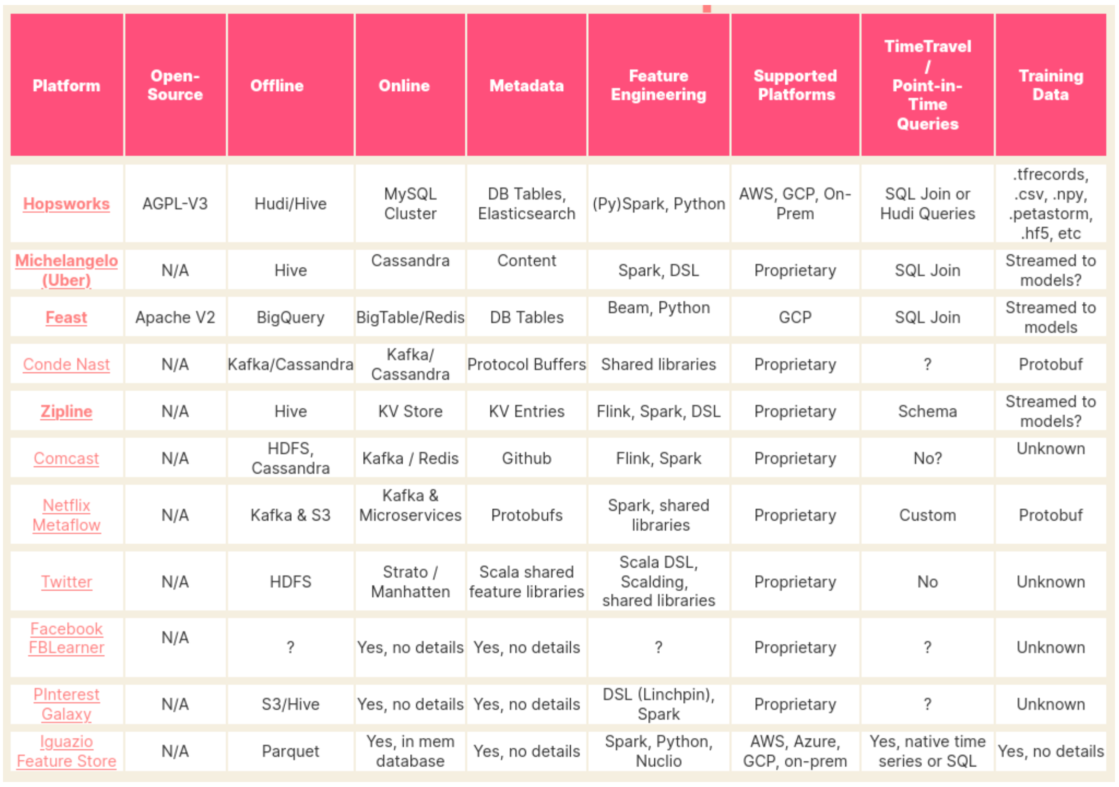


Figure 2.5: Hopsworks architecture [5]



Platform	Open-Source	Offline	Online	Metadata	Feature Engineering	Supported Platforms	TimeTravel / Point-in-Time Queries	Training Data
Hopsworks	AGPL-V3	Hudi/Hive	MySQL Cluster	DB Tables, Elasticsearch	(Py)Spark, Python	AWS, GCP, On-Prem	SQL Join or Hudi Queries	.tfrecords, .csv, .npy, .petastorm, .hf5, etc
Michelangelo (Uber)	N/A	Hive	Cassandra	Content	Spark, DSL	Proprietary	SQL Join	Streamed to models?
Feast	Apache V2	BigQuery	BigTable/Redis	DB Tables	Beam, Python	GCP	SQL Join	Streamed to models
Conde Nast	N/A	Kafka/Cassandra	Kafka/ Cassandra	Protocol Buffers	Shared libraries	Proprietary	?	Protobuf
Zipline	N/A	Hive	KV Store	KV Entries	Flink, Spark, DSL	Proprietary	Schema	Streamed to models?
Comcast	N/A	HDFS, Cassandra	Kafka / Redis	Github	Flink, Spark	Proprietary	No?	Unknown
Netflix Metaflow	N/A	Kafka & S3	Kafka & Microservices	Protobufs	Spark, shared libraries	Proprietary	Custom	Protobuf
Twitter	N/A	HDFS	Strato / Manhatten	Scala shared feature libraries	Scala DSL, Scalding, shared libraries	Proprietary	No	Unknown
Facebook FBLearner	N/A	?	Yes, no details	Yes, no details	?	Proprietary	?	Unknown
PInterest Galaxy	N/A	S3/Hive	Yes, no details	Yes, no details	DSL (Linchpin), Spark	Proprietary	?	Unknown
Iguazio Feature Store	N/A	Parquet	Yes, in mem database	Yes, no details	Spark, Python, Nuclio	AWS, Azure, GCP, on-prem	Yes, native time series or SQL	Yes, no details

Figure 2.6: Comparison between the most used Feature Stores [6]

2.1.6 Serving

Seldon Core Seldon Core is the most powerful Open Source platform for rapidly deploying machine learning models on Kubernetes. Seldon handles scaling to thousands of production machine learning models and provides advanced Machine Learning capabilities out of the box including advanced metrics (by using Prometheus, an Open Source monitoring tool), Request Logging, Explainers (by using its Open Source library, Alibi), Outlier Detectors, A/B tests, Canaries, Multi-Armed Bandits and more. Seldon Core can be integrated with Kubeflow to manage the deployment of the Machine Learning system from the pipeline orchestrator, Jenkins and JenkinsX for CI/CD and many other tools.

KFServing KFServing is a model deployment and serving toolkit, by Kubeflow, created to solve the core challenges about the model deployment. It "enables serverless inferencing on Kubernetes and provides performant, high abstraction interfaces for common Machine Learning frameworks like TensorFlow, XGBoost, scikit-learn, PyTorch, and ONNX" [36]. KFServing intends to provide an inference service, it allows data scientists to add transformers [2] and to add explainers to the core model server [3]. An autoscaler can also be further added to the KF-Serving service to watch traffic flow to the application and scale replicas based on configured metrics.

Other tools Another serving tool is BentoML. It is a flexible, high-performance framework for serving, managing, and deploying machine learning models. BentoML supports multiple frameworks (e.g. Tensorflow, PyTorch, Keras, XGBoost), cloud native deployment (e.g. with Docker, Kubernetes, AWS, Azure), an High-Performance online API serving and offline batch serving.

[2]Transformers allow data transformations of the request and response from the model; for example, a text model may need input words transformed into feature embedding vectors which are the raw input to the model.

[3]"Explainers allow model explanation methods to be attached to the service so an individual request/response from the model can be sent for providing human-understandable explanations. This allows users and auditors to better understand why a model is providing the predictions for certain inputs." [34]

Chapter 3

Design and implementation of a reference architecture for MLOps

Building an effective architecture, according to each different needs of a company, is crucial to get the best from MLOps.

Due to the plethora of tools, which are continuously arising, it may be very hard to select exactly the best technologies and infrastructures to use. First of all, it is necessary to understand the business process, how the models are developed and how they are put into production; by doing so a clear understanding of what tools and technologies better fit the business needs can be achieved. The MLOps space is not done yet and it is constantly evolving; it is important to have an architecture, from a technical point of view, which is very modular, in such a way that it will be possible to change the building blocks, the tools and the technologies at any time [37].

Chapter 2 offers a large variety of Open Source tools but, as pointed out, different solutions by the main cloud providers are available. Adopting Open Source instruments or building the MLOps architecture on a cloud solution is not a mutually exclusive choice, a mixed approach can be adopted. Open Source tools provide transparency and flexibility with respect to the cloud vendor since they typically are cloud-agnostic. Moreover, some tools are considered "standard de facto" at the moment, some noteworthy examples are Docker and Kubernetes; Open Source tools also allow a modular approach. On the other hand cloud solutions provide a full-stack environment, they ensure enterprise support and typically offer best-known tools with respect to specific Open Source instruments. However, proprietary solutions have different drawbacks: *"they reduce extendibility and transparency on a pipeline while enforcing heavy vendor lock-in. Further cloud providers services are often not a feasible solution for companies that work on regulatory software devices or software with user-privacy concerns; they require an MLOps solution that can be run cloud-agnostic and on-premises machines"* [38].

Existing cloud providers offer Machine Learning platforms such as AI Platform (Google Cloud), AzureML (Microsoft) and SageMaker (AWS) to build an MLOps architecture. *"The adoption of such Machine Learning platform depends on the cloud strategy of the organisation"* [29]. When an in-house hosted solution is preferred, using Open Source tools is a better choice (as they typically are cloud-agnostic).

3.1 Introducing an MLOps architecture in the business process

The process of introducing MLOps and building a successful Machine Learning system might be challenging for a company: it requires to change the approach of developing and deploying the system, it involves a lot of people in different teams (cf. Section 1.2.2), it needs to introduce new tools in the whole process and the tools to be introduced need to integrate with the existing enterprise systems, platform choices, pipeline strategy, and monitoring applications. MLOps should help the Machine Learning workflow, not inhibit it; thus, designing a good strategy to bring an MLOps approach in the business process is very important and it is crucial to focus on the change management it requires.

Two different approaches can be adopted; those procedures may also be interleaved, depending on the experience and the skills of the people involved, the customer's requirements and the infrastructure currently in use. The first method is based on the complexity of the tools, the second on the level of automation instead. Regardless of the approach, it is first essential to do a diagnosis of the current practices and processes used by the different teams, also by organising multiple interviews with the key stakeholders from the business, IT, Data Science and Ops teams. The most straightforward method concerns introducing MLOps technologies step-by-step, starting from the simplest tool: for example, MLFlow is very easy to use and it provides different benefits, from an MLOps point of view, by only using a Python API. The sooner the teams experience the benefits of MLOps best practices, the better; by doing so the people involved in the process familiarise themselves with the new method. Afterwards, the focus, when building an MLOps architecture with this approach, can shift on the tools which provide the highest value, possibly prioritising reproducibility and automation; by using Kubeflow, for example, reproducibility, validation and the focus on the process can be guaranteed (and, in addition, CI/CD and Continuous Training can be implemented by adopting, along Kubeflow itself, few other tools). A finer and neater approach consists of enhancing the level of automation in the process of building a Machine Learning system. As explained in Section 1.4.7, Google identifies three levels of automation [4]: in the first (level 0) the whole process is handled manually (without MLOps), level 1 adds the execution of the model training automatically and, the last stage, consists of introducing a full CI/CD system.

3.2 The reference architecture

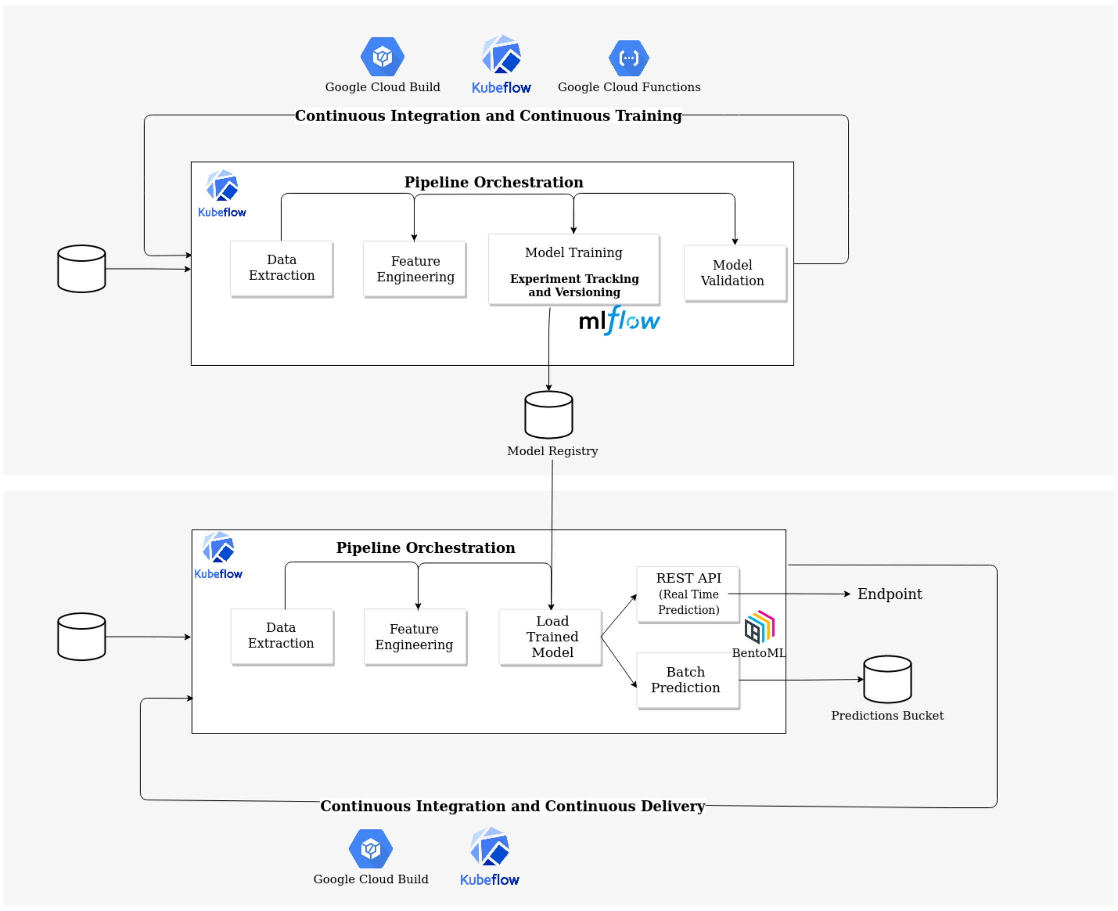


Figure 3.1: The designed and implemented MLOps architecture [7].

This section is intended to show the implementation of the reference MLOps architecture for this thesis and its benefits on the whole Machine Learning system; the architecture does not leverage in the considerations exposed in the previous section but an exploratory approach has been adopted. This implementation is the result of the internship at Data Reply; the goal of the company was to explore the different tools in the market and use them to build an MLOps architecture (by using preferably Open Source software).

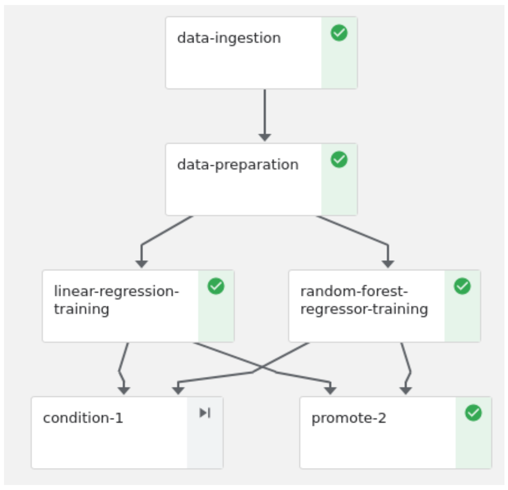


Figure 3.2: The DAG of the training pipeline on Kubeflow.

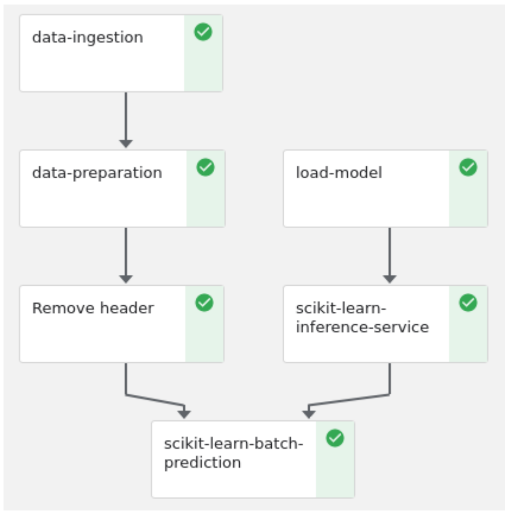


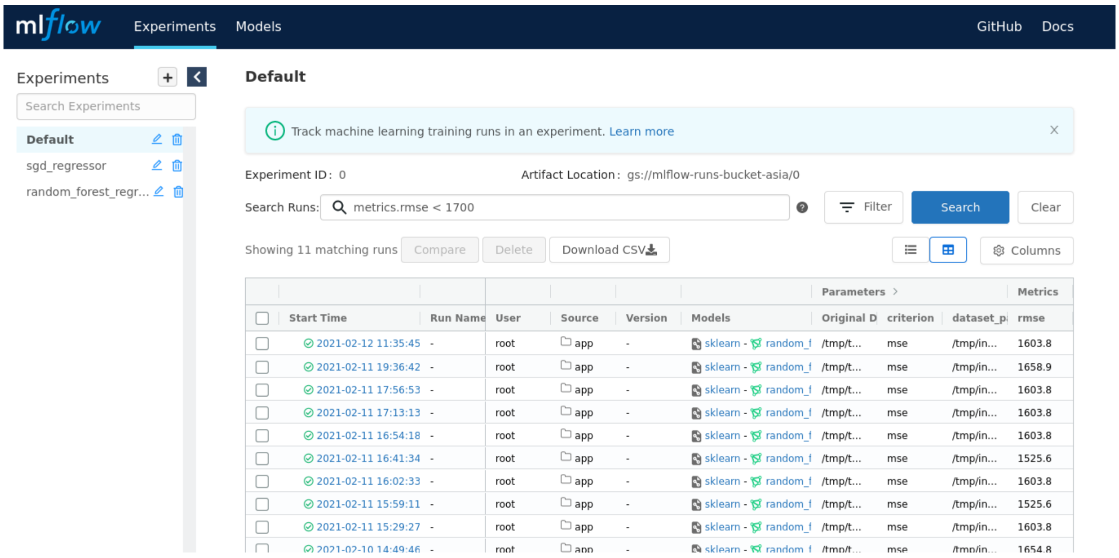
Figure 3.3: The DAG of the prediction pipeline on Kubeflow.

The project is built upon a Kaggle notebook [39] as use case example; though, here, the focus is on the whole architecture itself and not on the model only; in the Github repository [7] of the project detailed documentation about the architecture implementation is provided. The original notebook builds and trains three different models to forecast total German power consumption, on an hourly basis, with a lead time of 24 hours in the data; in this implementation two of the three models in the original notebook has been employed (SGD Regressor and Random Forest Regressor) and, instead of the German dataset, the Italian one has been used (available within the same notebook). Since the aim of this implementation is to build an MLOps architecture, instead of training a Machine Learning model, **the dataset** has a very simple structure; it is composed of a "start" column, an "end" column and a "load" column; "start" and "end" respectively contain the start time and the end time of the measured power consumption, the "load" column represents the power consumption itself in the range of time between "start" and "end".

Figure 3.1 shows the whole implemented architecture, which is composed of two different pipelines (a training pipeline and a prediction pipeline); the prediction pipeline was not in the original notebook and it has been implemented, in this project, in order to serve the trained models.

Implementation of the two pipelines The components of each pipeline are orchestrated together; this allows to foreground the whole process instead of focusing only on the model building. Figure 3.2 shows the DAG of the training pipeline implemented through Kubeflow, used in this project as pipeline orchestrator; the green tag, beside each component, represents the component itself has been run successfully. In order to implement the two pipelines, the original notebook has been split into multiple Docker containers, one for each component of the pipelines. This approach allows to build a modular architecture and also to reuse some components in both the pipelines (Data Extraction and Feature Engineering in Figure 3.1). The components of the training pipeline include: data ingestion, data preparation and feature engineering, model training and, the last step, is aimed to promote the model from “Staging” to “Production” according to its performances; the promotion of a model has been implemented comparing the results of the two trained models by using the *Conditions* mechanism of Kubeflow, which allows choosing a specific path in the DAG according to specific circumstances. The prediction pipeline (Figure 3.3), besides the components to extract and prepare the data, includes a component to load a pre-trained model and two different containers to build a batch and a real-time prediction service. Kubeflow allows to reuse components implemented by other people: the “remove-header” component in the prediction pipeline is an example of a “reusable component” and it removes the header from the dataset.

The implementation of both the pipelines, through different Docker containers, guarantees flexibility, reusability, portability, encapsulation and repeatability.



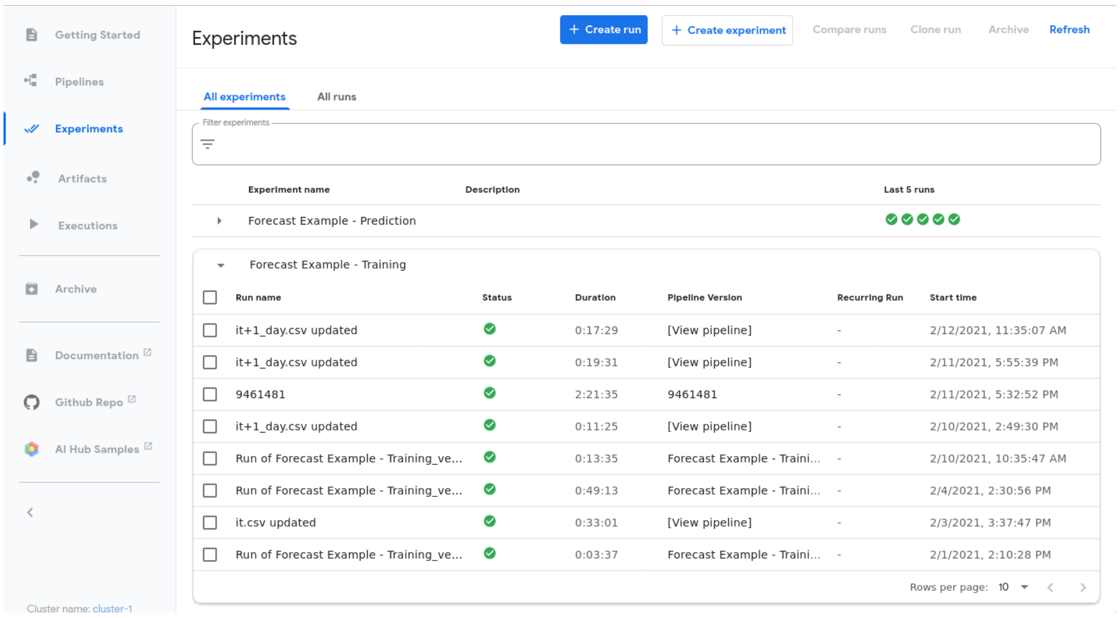
Experiment ID: 0   Artifact Location: gs://mlflow-runs-bucket-asia/0

Search Runs: metrics.rmse < 1700

Showing 11 matching runs

						Parameters			Metrics
Start Time	Run Name	User	Source	Version	Models	Original D	criterion	dataset_p	rmse
2021-02-12 11:35:45	-	root	app	-	sklearn - random_f	/tmp/t...	mse	/tmp/in...	1603.8
2021-02-11 19:36:42	-	root	app	-	sklearn - random_f	/tmp/t...	mse	/tmp/in...	1658.9
2021-02-11 17:56:53	-	root	app	-	sklearn - random_f	/tmp/t...	mse	/tmp/in...	1603.8
2021-02-11 17:13:13	-	root	app	-	sklearn - random_f	/tmp/t...	mse	/tmp/in...	1603.8
2021-02-11 16:54:18	-	root	app	-	sklearn - random_f	/tmp/t...	mse	/tmp/in...	1603.8
2021-02-11 16:41:34	-	root	app	-	sklearn - random_f	/tmp/t...	mse	/tmp/in...	1525.6
2021-02-11 16:02:33	-	root	app	-	sklearn - random_f	/tmp/t...	mse	/tmp/in...	1603.8
2021-02-11 15:59:11	-	root	app	-	sklearn - random_f	/tmp/t...	mse	/tmp/in...	1525.6
2021-02-11 15:29:27	-	root	app	-	sklearn - random_f	/tmp/t...	mse	/tmp/in...	1603.8
2021-02-10 14:49:46	-	root	app	-	sklearn - random_f	/tmp/t...	mse	/tmp/in...	1654.8

Figure 3.4: Successful runs of the model training experiments, tracked through MLFlow.



Experiment name	Description	Last 5 runs
Forecast Example - Prediction		✓✓✓✓✓

Forecast Example - Training

Run name	Status	Duration	Pipeline Version	Recurring Run	Start time
it+1_day.csv updated	✓	0:17:29	[View pipeline]	-	2/12/2021, 11:35:07 AM
it+1_day.csv updated	✓	0:19:31	[View pipeline]	-	2/11/2021, 5:55:39 PM
9461481	✓	2:21:35	9461481	-	2/11/2021, 5:32:52 PM
it+1_day.csv updated	✓	0:11:25	[View pipeline]	-	2/10/2021, 2:49:30 PM
Run of Forecast Example - Training_ve...	✓	0:13:35	Forecast Example - Traini...	-	2/10/2021, 10:35:47 AM
Run of Forecast Example - Training_ve...	✓	0:49:13	Forecast Example - Traini...	-	2/4/2021, 2:30:56 PM
it.csv updated	✓	0:33:01	[View pipeline]	-	2/3/2021, 3:37:47 PM
Run of Forecast Example - Training_ve...	✓	0:03:37	Forecast Example - Traini...	-	2/1/2021, 2:10:28 PM

Rows per page: 10

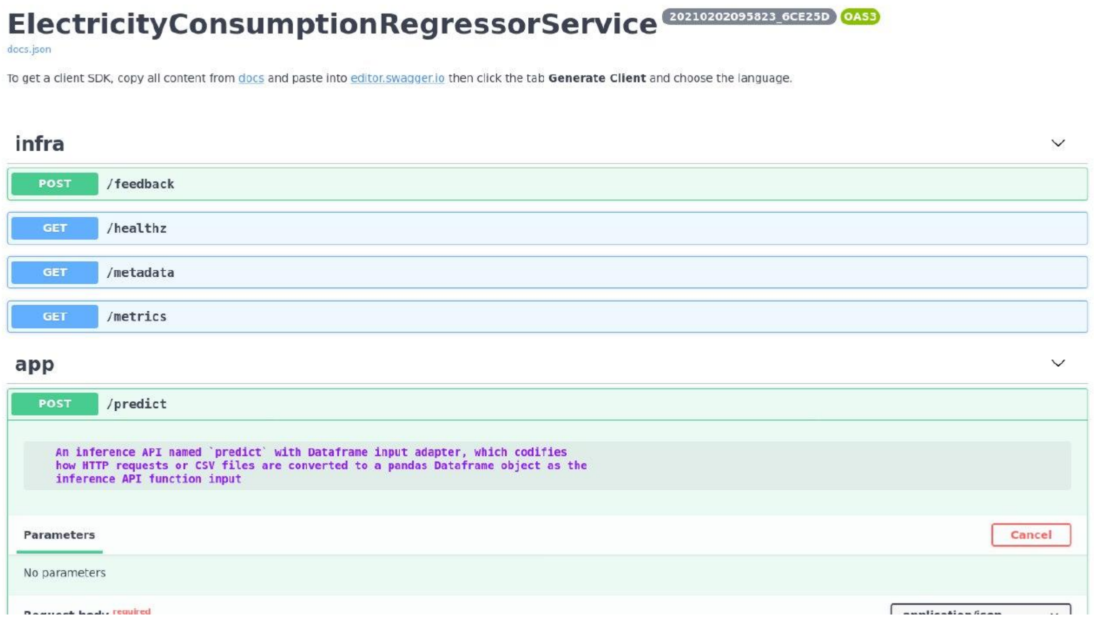
Figure 3.5: Runs of the training and predictions pipelines in Kubeflow.

The data flow Within the training pipeline the data, first of all, are ingested and appended to the existing data (in the "data-ingestion" component in Figure 3.2), then the dataset is transformed and prepared in such a way as to be processed by the model ("data-preparation" component in Figure 3.2) by performing feature engineering; afterwards the data pass to the *Model Training and Validation* component (which, in Figure 3.2 could be either the "linear-regression-training" or the "random-forest-regressor-training" component).

In the prediction pipeline data are ingested and prepared in the same way as the training pipeline, thereafter they feed either the *REST API* or the *Batch prediction* components; the output produced by the *Batch Prediction* component is finally saved on the prediction bucket.

Implementation of Models Experiment Tracking and Versioning The model training component includes experiment tracking and versioning, implemented by using MLFlow. The component, in addition to build and train the model, also logs the parameters (cf. Figure 2.2), the metrics (cf. Figure 3.4) and the artefacts produced by the model itself; through MLFlow it also tracks the model versions (cf. Figure 2.3) and stores them in a Model Registry. MLFlow is also used, in this project, to promote a version of a trained model for a "Production" usage and to load it in the prediction pipeline (in such a way as to be used for batch or real-time analysis). Figure 3.4 shows some of the successful runs of the experiments tracked by MLFlow.

By tracking the whole training, in this architecture, all the experiments can be easily reproduced, versioned, compared with each other and executed in the same, logged, environment.


ElectricityConsumptionRegressorService 20210202095823_6CE25D OAS3

docs.json

To get a client SDK, copy all content from docs and paste into editor.swagger.io then click the tab **Generate Client** and choose the language.

infra

POST /feedback

GET /healthz

GET /metadata

GET /metrics

app

POST /predict

```
An inference API named `predict` with Dataframe input adapter, which codifies
how HTTP requests or CSV files are converted to a pandas Dataframe object as the
inference API function input
```

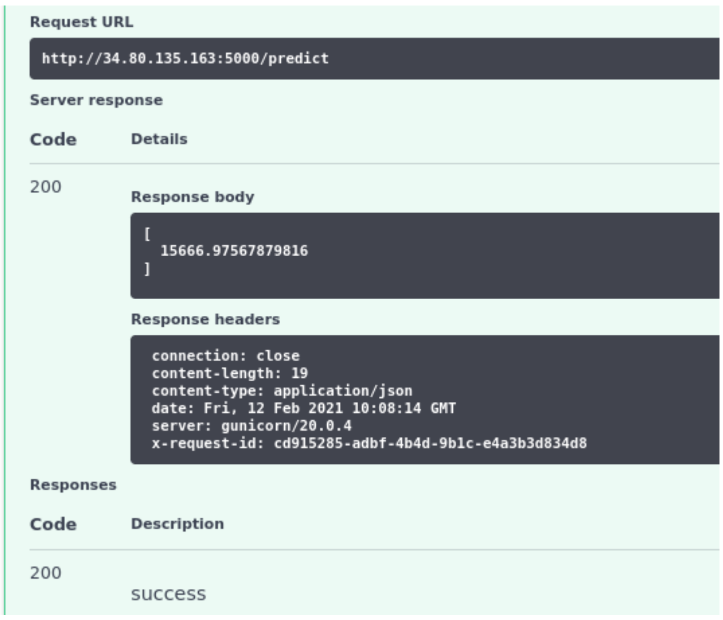
Parameters Cancel

No parameters

Figure 3.6: OpenAPI specifications of the BentoML service.

Implementation of Automation, Continuous Integration, Continuous Delivery and Continuous Training The training and the prediction pipelines can be executed automatically according to a run schedule. Moreover, when the code in the repository changes (for example as a result of a commit or a merge) or new data are available, the two pipelines are automatically run. These two features, Continuous Integration and Continuous Training, enable automation and they have been implemented through Kubeflow and, respectively, Google Cloud Build and Google Cloud Functions. Figure 3.5 shows some runs of the training and the prediction pipelines on Kubeflow, the green tag in the "Status" column represents the run completed successfully; in the "Run name" column the runs with the commit hash are those triggered by Google Cloud Build and, the runs with the updated dataset name, those triggered by Google Cloud Functions.

Despite Seldon Core and KFServing are the main tools on the landscape of Open Source serving tools, in this project both the batch and the real-time prediction service have been implemented through BentoML. As mentioned in Section 2.1.6 BentoML is an emerging high-performance Open Source framework for



Request URL

```
http://34.80.135.163:5000/predict
```

Server response

Code	Details
200	Response body

```
[
  15666.97567879816
]
```

Response headers

```
connection: close
content-length: 19
content-type: application/json
date: Fri, 12 Feb 2021 10:08:14 GMT
server: gunicorn/20.0.4
x-request-id: cd915285-adbf-4b4d-9b1c-e4a3b3d834d8
```

Responses

Code	Description
200	success

Figure 3.7: Successful response of a prediction through the BentoML ReST API.

serving, managing, and deploying Machine Learning models. This choice is due to the effectiveness and the easiness of use of BentoML. This tool allows to build a REST API (by also providing an OpenAPI graphic interface and its specifications, cf. Figure 3.6) and a prediction service in a very simple fashion and with few lines of code. Figure 3.7 shows a successful response of a prediction through the BentoML ReST API.

3.3 Deployment of the architecture

The whole project is designed and implemented to run on Google Cloud Platform and it uses both Open Source software and tools from the AI Platform by Google. Since all the three Open Source tools used in this project (MLFlow, Kubeflow and BentoML) are platform-agnostic they can be executed on any other cloud platform.

Google Cloud Platform is the high-performance infrastructure, by Google, for cloud computing, data analytics and Machine Learning. The technologies adopted in this project, from Google Cloud Platform, are:

- Google Cloud Storage
- Google Cloud Functions

- Google Cloud Build
- Google Container Registry
- Kubeflow (which, in Google Cloud Platform, is called AI Platform Pipelines).

Google Cloud Storage Google Cloud Storage is a service for storing objects in Google Cloud Platform. In Google Cloud Storage the data are held by basic containers, called "Buckets".

Google Cloud Functions Cloud Functions is a lightweight solution to create single-purpose, stand-alone functions that respond to Cloud events without the need to manage a server or runtime environment; namely a tool for creating event-driven applications according to the "Function as a Service" (FaaS) paradigm.

In this project a Function is used to provide Continuous Training: whenever a new file is uploaded to the bucket, where the training dataset is stored, the Function triggers a new run of the Kubeflow (training) pipeline. In practice, inside the training pipeline code, a Python function is defined: this is the Cloud Function to be triggered. This generic function logs relevant data when a file is changed, compiles the training Kubeflow Pipeline and runs it.

Google Cloud Build Google Cloud Build is the serverless CI/CD tool in Google Cloud Platform. It allows to import sources from Cloud Source Repositories,

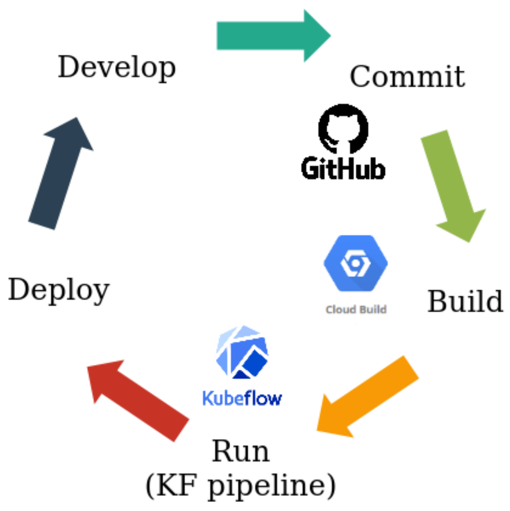
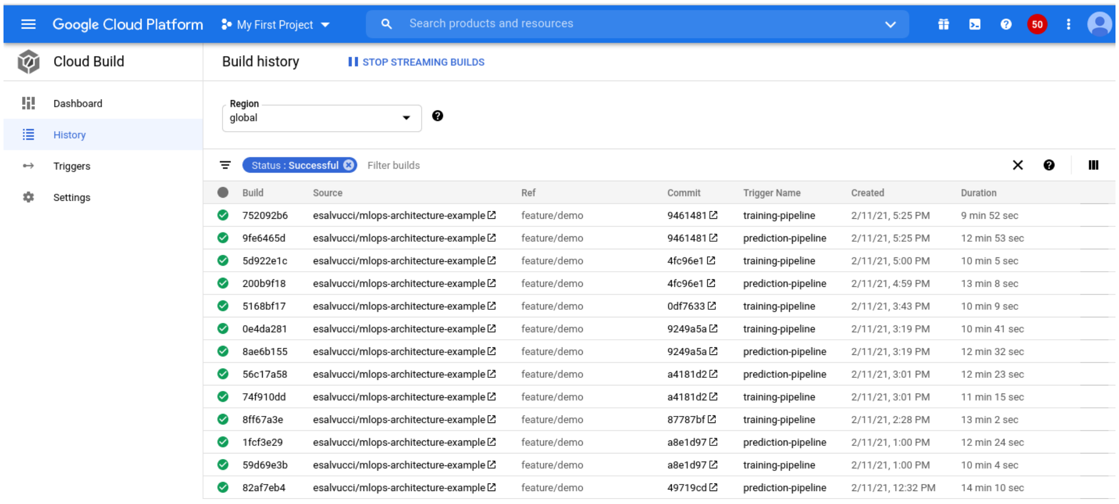


Figure 3.8: The sequence of the steps performed in the CI/CD flow.



Region: global

Status : Successful

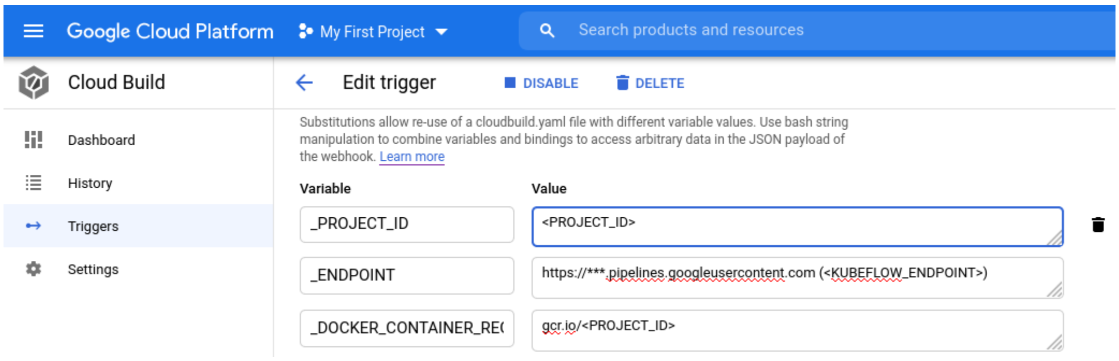
	Build	Source	Ref	Commit	Trigger Name	Created	Duration
✓	752092b6	esalvucci/mlops-architecture-example	feature/demo	9461481	training-pipeline	2/11/21, 5:25 PM	9 min 52 sec
✓	9fe6465d	esalvucci/mlops-architecture-example	feature/demo	9461481	prediction-pipeline	2/11/21, 5:25 PM	12 min 53 sec
✓	5d922e1c	esalvucci/mlops-architecture-example	feature/demo	4fc96e1	training-pipeline	2/11/21, 5:00 PM	10 min 5 sec
✓	200b9f18	esalvucci/mlops-architecture-example	feature/demo	4fc96e1	prediction-pipeline	2/11/21, 4:59 PM	13 min 8 sec
✓	5168bf17	esalvucci/mlops-architecture-example	feature/demo	0df7633	training-pipeline	2/11/21, 3:43 PM	10 min 9 sec
✓	0e4da281	esalvucci/mlops-architecture-example	feature/demo	9249a5a	training-pipeline	2/11/21, 3:19 PM	10 min 41 sec
✓	8ae6b155	esalvucci/mlops-architecture-example	feature/demo	9249a5a	prediction-pipeline	2/11/21, 3:19 PM	12 min 32 sec
✓	56c17a58	esalvucci/mlops-architecture-example	feature/demo	a4181d2	prediction-pipeline	2/11/21, 3:01 PM	12 min 23 sec
✓	74f910dd	esalvucci/mlops-architecture-example	feature/demo	a4181d2	training-pipeline	2/11/21, 3:01 PM	11 min 15 sec
✓	8ff67a3e	esalvucci/mlops-architecture-example	feature/demo	87787bf	training-pipeline	2/11/21, 2:28 PM	13 min 2 sec
✓	1fcf3e29	esalvucci/mlops-architecture-example	feature/demo	a8e1d97	prediction-pipeline	2/11/21, 1:00 PM	12 min 24 sec
✓	59d69e3b	esalvucci/mlops-architecture-example	feature/demo	a8e1d97	training-pipeline	2/11/21, 1:00 PM	10 min 4 sec
✓	82af7eb4	esalvucci/mlops-architecture-example	feature/demo	49719cd	prediction-pipeline	2/11/21, 12:32 PM	14 min 10 sec

Figure 3.9: Runs on Google Cloud Build, triggered by a commit or a merge in the code repository.

Github or Bitbucket and, then, build each component according to a build configuration file (cloudbuild.yaml) and produce artefacts such as Docker containers.

In this architecture, CI/CD is implemented through Kubeflow and Google Cloud Build together (Figure 3.8). Each component of the Kubeflow Pipeline is built by Google Cloud Build, which is triggered by a specific event on the Github repository (e.g. a commit or a merge). When the event occurs, the Kubeflow Pipeline is compiled and run. In such a way, whenever a piece of code changes in the Github repository, each component of the Kubeflow Pipeline is built up and the Kubeflow Pipeline itself is automatically executed.

A very powerful feature, in Google Cloud Build, is *Substitution*. Cloud Build allows you to use variables in the configuration file and define their actual value be-



Substitutions allow re-use of a cloudbuild.yaml file with different variable values. Use bash string manipulation to combine variables and bindings to access arbitrary data in the JSON payload of the webhook. Learn more

Variable	Value
_PROJECT_ID	<PROJECT_ID>
_ENDPOINT	https://***.pipelines.googleusercontent.com (<KUBEFLOW_ENDPOINT>)
_DOCKER_CONTAINER_RE(	gcr.io/<PROJECT_ID>

Figure 3.10: Example of substitution variables in Google Cloud Build.

fore each individual run; this characteristic is helpful for variables whose value isn't known until build time. These variables include $COMMIT_SHA, $REPO_NAME, $BRANCH_NAME, $TAG_NAME. Other non-trigger-based variables are $PROJECT_ID and $BUILD_ID. In this project Substitutions are used, specifically, to tag each Kubeflow run, triggered by Google Cloud Build, with the corresponding git commit hash. Figure 3.10 shows an example of Substitutions used in this project; Figure 3.9, instead, shows some successful run on Google Cloud Build, triggered as a consequence of a commit or a merge in the code repository.

Google Container Registry Google Container Registry is a container image registry that runs on Google Cloud Platform; in this project, it is used to push the Docker images of each Kubeflow component and pull them from the training or prediction pipeline.

3.4 The training and the prediction pipelines in detail

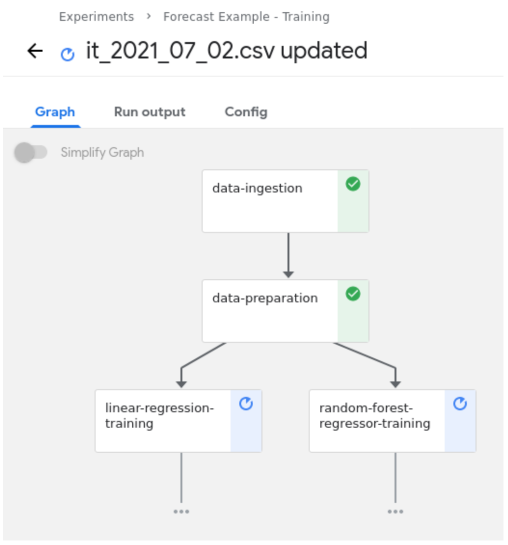

Experiments › Forecast Example - Training
it_2021_07_02.csv updated
Graph
Run output
Config
Simplify Graph
data-ingestion
data-preparation
linear-regression-training
random-forest-regressor-training

Figure 3.11: The DAG of the training pipeline on Kubeflow, triggered by Google Cloud Functions as consequence of a file upload (or update), while running the model traininig components.

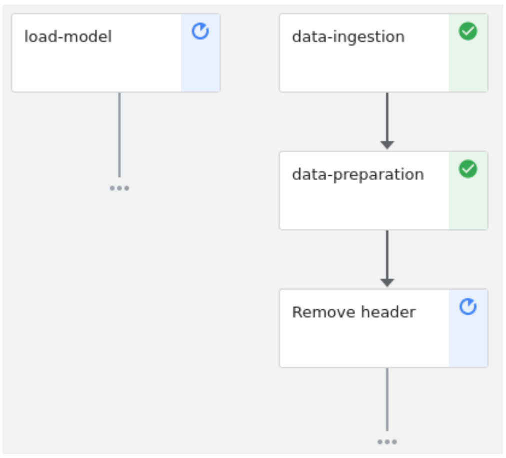


Figure 3.12: The DAG of the prediction pipeline on Kubeflow while running both the load-data and remove-header components.

Listing 3.1: The definition of the data-ingestion Kubeflow component

```
def __data_ingestion_step(bucket_name):
    return kfp.dsl.ContainerOp(
            name='data_ingestion',
            image=os.environ['
               DOCKER_CONTAINER_REGISTRY_BASE_URL'] +
                  '/' +
                  os.environ['PROJECT_NAME'] +
                  '/' +
                  os.environ['DATA_INGESTION'] +
                  ':' +
                  os.environ['TAG'],
            arguments=['--bucket_name', bucket_name],
            file_outputs={'dataset_path':
                          '/tmp/dataset.csv'}
    )
```

In the previous section a generic description of the training and the prediction pipelines was provided; in this section will be explained, in more detail, the execution of the two pipelines. The first one is aimed at build and train two different models; the second one, on the other hand, is aimed at building two different services that handle batch or real-time predictions. Figure 3.11 and Figure 3.12 shows the two running pipelines.

The training pipeline The training pipeline (Figure 3.2) builds and trains an SGD Regressor and a Random Forest Regressor. The pipeline can be triggered manually, according to a scheduled run, by Google Cloud Build or by Google Cloud Function. When new data are available, within the bucket containing the dataset, the pipeline is automatically triggered by Google Cloud Function; Figure 3.13 shows the logs of the Function triggered as a consequence of a new file upload or update and Figure 3.11, instead, the running pipeline triggered by the Function itself. To run the pipeline, through Google Cloud Functions, the latest version of the Docker container of each component is used. When the pipeline is triggered by Google Cloud Build, as a consequence of a commit or a merge in the code repository, each component is built up in a new Docker container (according to the new code) and, then, the pipeline starts its run.

The pipeline is orchestrated through Kubeflow. Listing 3.1 shows how a Kubeflow component is defined (the data-ingestion component in the example): the

Listing 3.2: The kfp-cli command used to run programmatically the Kubeflow pipeline

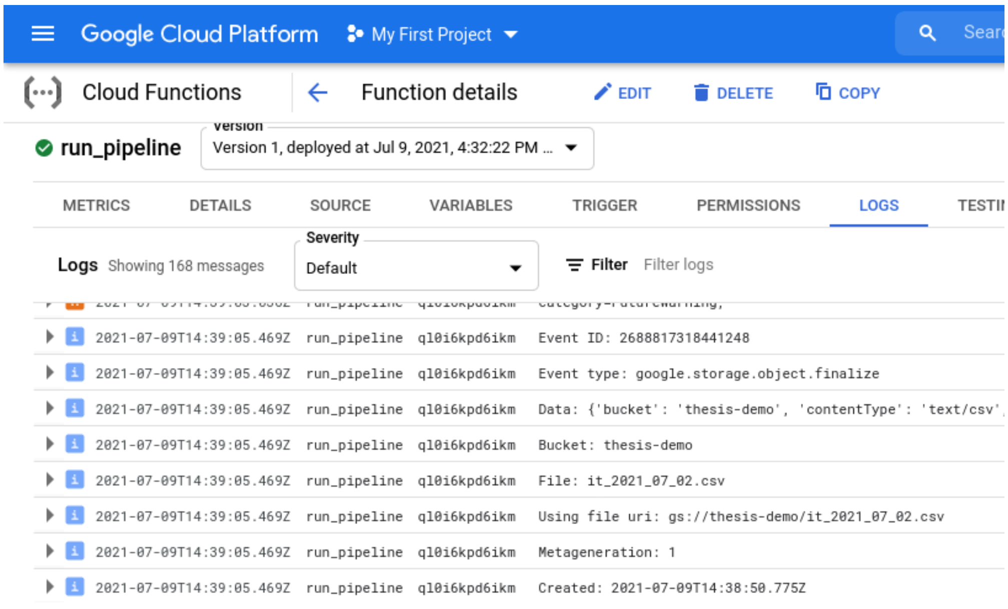
```
kfp --endpoint $_ENDPOINT run submit
              -e "${_PIPELINE_NAME}"
              -r ${SHORT_SHA}
              -p $(kfp --endpoint $_ENDPOINT pipeline
                 list | grep -w "${_PIPELINE_NAME}"  |
                   grep -E -o -e "([a-z0-9]){8}-([a-z0
                      -9]){4}-([a-z0-9]){4}-([a-z0-9])
                      {4}-([a-z0-9]){12}")
```

function returns a ContainerOp, which takes in input the name of the component, the docker image to be retrieved to run the component and a list of arguments (according to the type of argument both a simple parameter or a file can be passed); the parameter "file_outputs" defines the name of the artefacts produced in output by the component itself, which will be passed to the next component of the pipeline. To run the pipeline through Google Cloud Build the Kubeflow's kfp-cli command is employed in different ways: Listing 3.2 shows the command to run the new pipeline on Kubeflow.

The pipeline starts with the "data-ingestion" component; this component gets each file, from the dataset bucket, and appends it to the others. The dataset, merged in a single csv file, is passed to the "data-preparation" component which performs Feature Engineering: time features (month, weekday and hour), national holiday features and lag features (load data with a lag value ranging from 24 to 48 hours) are added, then one-hot encoders, on categorical features (time features), and a standard scaler, on numerical features (lag features), are applied.

The dataset (including all the new features) is passed to both the model training components: "linear-regression-training" and "random-forest-regressor-training" in Figure 3.2; the first component trains an SGD Regressor and, the second one, a Random Forest Regressor. The experiments performed on both the models are tracked and versioned through MLFlow (including the parameters, the metrics and the artefacts); this approach allows to easily reproduce the experiments and compare them. The built models are saved on the MLFlow Model Registry, in such a way that they can be easily loaded in the prediction pipeline. In both, the Model Training components Model Validation is also performed.

The last component of the training pipeline, "promote" in Figure 3.2, makes use of *Conditions*, a mechanism of Kubeflow that allows choosing a path in the DAG according to specific circumstances; in the "promote" component a "condition"



run_pipeline Version: Version 1, deployed at Jul 9, 2021, 4:32:22 PM ...

Logs Showing 168 messages Severity: Default

```
2021-07-09T14:39:05.469Z  run_pipeline  ql0i6kpd6ikm  Event ID: 2688817318441248
2021-07-09T14:39:05.469Z  run_pipeline  ql0i6kpd6ikm  Event type: google.storage.object.finalize
2021-07-09T14:39:05.469Z  run_pipeline  ql0i6kpd6ikm  Data: {'bucket': 'thesis-demo', 'contentType': 'text/csv',
2021-07-09T14:39:05.469Z  run_pipeline  ql0i6kpd6ikm  Bucket: thesis-demo
2021-07-09T14:39:05.469Z  run_pipeline  ql0i6kpd6ikm  File: it_2021_07_02.csv
2021-07-09T14:39:05.469Z  run_pipeline  ql0i6kpd6ikm  Using file uri: gs://thesis-demo/it_2021_07_02.csv
2021-07-09T14:39:05.469Z  run_pipeline  ql0i6kpd6ikm  Metageneration: 1
2021-07-09T14:39:05.469Z  run_pipeline  ql0i6kpd6ikm  Created: 2021-07-09T14:38:50.775Z
```

Figure 3.13: The logs of Google Cloud Functions when a new file in the bucket is uploaded (or updated).

compares the metrics of the two, trained, models and tags the best one as available to be used in production.

The prediction pipeline The prediction pipeline (Figure 3.3) builds two different services to handle real-time and batch predictions; the pipeline can be triggered manually, by a scheduled run or by Google Cloud Build (as a consequence of a commit or a merge).

Thanks to the modular implementation of the training pipeline, the prediction pipeline can reuse both "data-ingestion" and "data-preparation" components. In this pipeline Kubeflow is used as an orchestration tool as in the training pipeline; this tool also allows to employ components designed and implemented by other people: this kind of components is called "reusable components"; the "remove-header" component, which is a "reusable component", removes the header from the dataset.

In a parallel path (cf. Figure 3.12), the model to be used for the prediction (the one with the tag "Production") is loaded through MLFlow within the "load-model" component. The prediction pipeline includes a service to perform real-time predictions and another to perform batch predictions: "scikit-learn-inference-service" is the component which produces an artefact, deployed by using BentoML, to handle real-time predictions; "scikit-learn-batch-prediction", instead, gets the artefact produced by the "scikit-learn-inference-service" and employs it to perform a batch prediction on the transformed dataset.

3.5 Code publication

The code of the implementation, of the reference architecture, is available on Github [7] and is published as Free Software, under GPLv3 licence. The repository aims to provide the implementation of the architecture and a detailed explanation of how the tools have been employed; the advantages on the Machine Learning workflow have been also underlined within the project.

The *doc* directory contains deep documentation of the usage of each tool employed within the architecture. In the *demo* folder a concise explanation on how to run the whole project is provided; in the same directory the dataset used in the project can be also found. The *components* directory contains the code of each component of both the two Kubeflow pipelines; the folder of each component is constituted of:

- A *Dockerfile*, to build the corresponding Docker image.
- A bash script, used to build the Docker image of the component and push it into the Container Registry.
- The *src* folder, which contains the code of the component.
- The requirements.txt file, used to install all the dependencies of the component.

The *prediction_pipeline* and *training_pipeline* folders contain the code used to define the two Kubeflow pipelines. In both the directories a cloudbuild.yaml file is also included: it defines the steps to be executed by Google Cloud Build to build all the components, compile the Kubeflow pipeline and run it. *prediction_pipeline training_pipeline* also contains a Dockerfile: it is used to build a Docker image with *kfp-cli*, the Kubeflow's command line tool (which is employed in the cloudbuild.yaml file). The *main.py* file contains the implementation of the Kubeflow pipeline. Furthermore, the *.env.yaml* file, used by Google Cloud Function, contains the environment variables needed to compile the Kubeflow pipeline.

Chapter 4

MLOps use cases and scenarios

MLOps is a very recent methodology, nevertheless it is rapidly taking hold in business contexts thanks to the benefits it has on putting a model into production; as explained in Chapter 1, MLOps can help to significantly reduce the time to deploy models, allowing more flexibility on updating them. In the following will be discussed different use cases in which MLOps is currently used by different companies of different business domains; a generic scenario related to the Covid-19 pandemic will be also provided. All the architectures exposed below share a common trait: they are built up according to the specific needs of each company.

Covid-19 pandemic Modern business applications leverage Machine Learning and Deep Learning models to analyze real-world and large-scale data, to predict, or to react intelligently to events; however, data change according to the environment and the events, causing Concept Drift. As already mentioned, Concept Drift is a challenging problem in Data Science: it may happen due to changes in consumer preferences, technological innovations, catastrophic events, etc. Covid-19 pandemic is a very clear example of this problem; "*the pandemic disrupted many supply chains because demand planning models weren't updated frequently enough to account for the quickly emerging "new normal" as the pandemic itself began*" [24].

Covid-19 caused a huge change in the data used to make predictions. The unexpected first lockdown, which has expanded becoming worldwide, day by day, deeply impacted the sales of every kind of market and, consequently, on their data. The Covid-19 pandemic caused Machine Learning models across many industries to go haywire because of rapidly changing conditions; moreover, due to the long time to get a model in production, it was difficult to overcome the Concept Drift caused by the pandemic itself.

"*Investing in MLOps allows organisations, and their Machine Learning solutions in production, to be more resilient to external volatile events, like rapid mar-*

ket landscape changes, regulatory changes, and other unforeseen external events like the Covid-19 pandemic" [40]. MLOps is primarily aimed at reducing the time to get a model in production as much as possible. One of the most important features of MLOps, to face Concept Drift, is Continuous Training, since it helps to be fastly resilient to unexpected events which cause a change in the data. MLOps speeds up the development, deployment, and management of models, thus enabling the creation of applications that can rapidly adapt to changes in the environment. "*Using MLOps automation, businesses can monitor and detect changes that impact their AI models, make swift changes to their AI applications and get new solutions to market faster and in a much more agile way*" [41].

AstraZeneca AstraZeneca, multinational pharmaceutical company, leverage Machine Learning and Deep Learning techniques to accelerate drugs discovery. Its platform, called *Augmented Drug Design*, helps chemicals develop drugs faster by using Machine Learning and other techniques, alongside the work performed in chemical laboratories; this approach allows to save both a lot of time and money.

"MLOps plays a key part in AstraZeneca's mission to reduce the research phase of the drug discovery cycle by half, from 24 months to 12 months, by 2025" [37] says Adrian Rossall, head of Augmented Drug Design, at the Seldon 1.0 launch event. MLOps helps AstraZeneca deploying models faster, monitoring and retraining them. AstraZeneca makes use of scientifically aware and industry-specific tools, for this reason it needed a customizable solution; other requirements to build an MLOps architecture, for the company, were flexibility and scalability. In 2019 AstraZeneca started using Seldon on top of Kubernetes. Seldon simplified and enhanced model deployment, Kubernetes allowed to scale up and down, granted flexibility and allowed the company to deploy models faster; "*MLOps really helped to accelerate the whole pipeline*", says Adrian Rossal [37].

Netflix Netflix makes extensive use of Machine Learning across a lot of areas in their product and deploys thousands of models; this helps to personalize the experience of the customers and the content necessary to offer them an optimal experience. The business problem, for Netflix, is to estimate the size of the audience, every day, of a show in the months leading up to the show's launch. This is important for a variety of reasons, including prioritization, allocation of resources etc. Both accuracy and timeliness are crucial, because if predictions aren't accurate and fast then they're not useful and, at the same time, they lose the opportunity to make decisions based on them. A project typically starts with the exploration of the data and looking for correlations among them; this phase can take between two to four weeks. The next stage involves building and identifying the candidate model to solve the business problem; this usually takes about six to eight weeks.

Then the model needs to be shipped into production, which concerns different tasks and which requires from 12 to 14 weeks [8].

Metaflow is the Open Source Machine Learning infrastructure, originally developed at Netflix, to boost the productivity of data scientists. Metaflow can help for rapid prototyping and for a fast deployment of the models, reducing the time to get a model in production to less than 12 weeks [8]. Metaflow enables collaboration, offers first-class support for prototyping and deployment, allows straightforward scalability, provides specific data tooling and is designed to make operational issues easy to diagnose and fix [42]. Metaflow can be thought of as a wrapper of well-known tools: XGBoost, PyTorch, Tensorflow as Machine Learning libraries, Pandas for Feature Engineering, Jupyter as a collaborative tool; Meson, a workflow management tool, to provide job scheduling and task isolation with Titus and Apache Mesos, Spark as query engine and Amazon S3 as Data Lake. Figure 4.1 shows the stack of the tools included in Metaflow.

Compared to the architecture described in Chapter 3 Metaflow is a general-purpose platform; it makes use of Airflow instead of Kubeflow and it includes query engine and data lake tools, missing in the architecture exposed in the previous chapter. Moreover, Metaflow does not include any tool aimed to track the experiments for the model training.

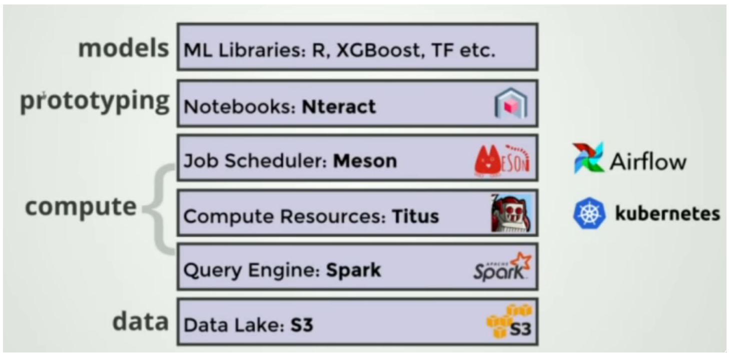


Figure 4.1: Stack of the technologies included in Metaflow [8].

Bank Itaú Unibanco *"Itaú Unibanco is the largest private sector bank in Brazil, with a mission to put its customers at the centre of everything they do as a key driver of success"* [9]; to deal with this goal the bank built *Itaú Virtual Assistant*, a digital customer service tool that uses Natural Language Processing to understand customer questions and respond in real-time. To help continually improve and evolve Itaú Virtual Assistant the bank needed an efficient strategy for the deployment of Machine Learning models; hence the Machine Learning team designed a CI/CD pipeline, based on Kubeflow, on Google Cloud Platform. *"For the Itaú*

Virtual Assistant project, two business requirements were essential: the ability to have multiple models in production (whether using different techniques or models trained using distinct data), and the ability to retrain the production model with new data" [9].

The architecture was designed by using Open Source tools, including Kubeflow, Kubernetes, Seldon Core, Docker, and Git. The goal was to have a single overall solution that could be deployed on Google Cloud Platform or on-premises (for example Origin, the Open Source version of RedHat OpenShift), according to the needs and restrictions of each team inside the company. Figure 4.2 shows the architecture built up by Unibanco.

Unibanco's architecture is very similar to the architecture exposed in Chapter 3: it includes a CI/CD pipeline (which, instead of Google Cloud Build, is implemented through Jenkins) and, as the architecture in the previous chapter, it implements a model training pipeline by using Kubeflow; Seldon Core is used for the model serving instead of BentoML. Despite these differences, the reference architecture for this thesis would fit Unibanco's requirements.

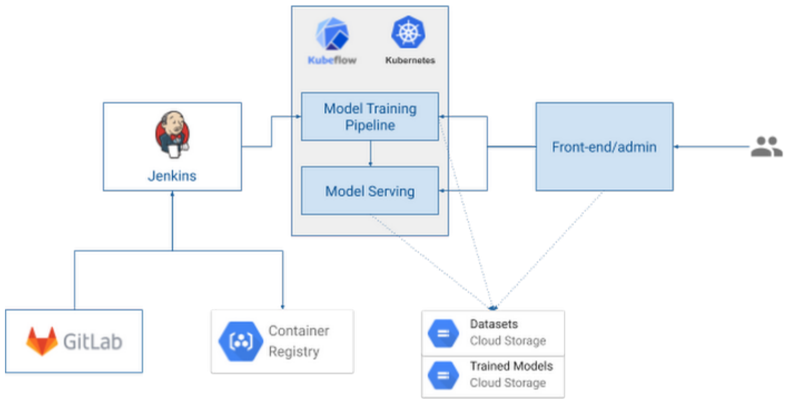


Figure 4.2: CI/CD architecture of Itau Unibanco [9].

GreenSteam GreenSteam is a company that provides software solutions for optimizing vessel performance, to save fuel and reduce emissions. Even though GreenSteam has already built several Machine Learning products in the past, which helped some major shipping companies make informed performance optimization decisions, in 2019 the need for a renewal of the process of building Machine Learning models emerged: "*We knew our Machine Learning operations needed to grow with the company*" [10]. For this reason, the company decided to start from scratch and rethink its entire Machine Learning infrastructure.

The first step involved switching from Jupyter notebooks to Python packages, versioned on git repositories. Subsequently, the company faced reproducibility

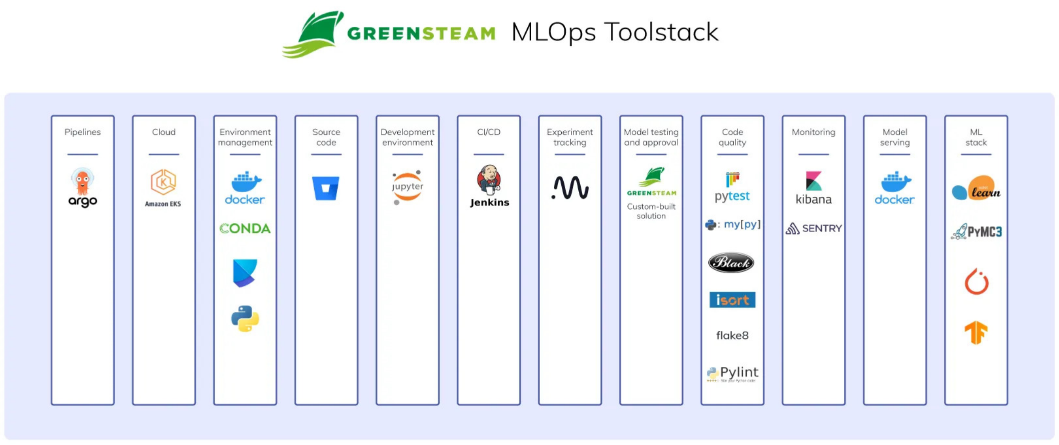


Figure 4.3: MLOps tools stack used by GreenStream [10].

issues; while the setup of the environments, in the different laptops, was similar it was never the same: GreenStream started using conda, but it did not solve the problem; "*Docker helped with the problem and it enabled GreenSteam to have a unified setup*" [10]. Dealing with Docker containers also helped GreenStream to build a Continuous Integration pipeline by using Jenkins. Through Docker the company paved the way to move from monoliths to microservices, orchestrated by using Argo (Open Source pipeline orchestrator). GreenStream also needed a custom model serving solution: SageMaker and Kubeflow have been looked at as solutions, but the tools did not meet the needs; FastAPI turned out to be the best tool for the company requirements. The last step of rethinking the Machine Learning infrastructure, for GreenStream, involved tracking the experiments, their version and their metrics: Neptune was used as a tool. Figure 4.3 shows the full technologies stack used by GreenStream for its MLOps infrastructure.

The architecture exposed in Chapter 3 would probably fit GreenStream's requirements. The architecture built up by the company, however, differs in the employed tools: Argo is used instead of Kubeflow to orchestrate the pipelines, AWS is used as cloud platform, Jenkins to implement CI/CD (instead of Kubeflow and Google Cloud Build) and Neptune as Experiment Tracking tool, instead of MLFlow.

Uber Uber employs Machine Learning to make data-driven decisions; it not only enables services such as ridesharing (destination prediction, driver-rider pairing, ETA prediction, etc) but also financial planning and other core business needs. Machine Learning solutions are also implemented in some of Uber's other businesses such as UberEATS, UberPool, and Uber's self-driving car division [43].

Uber faced, in the past, different challenges with building and deploying Machine Learning models: there were no systems to build reliable, uniform and reproducible pipelines and there was neither a standard place to store the results of training experiments and compare them together. "*We were starting to see signs of many of the Machine Learning ant-patterns documented by Scully et al. [1]*" [11].

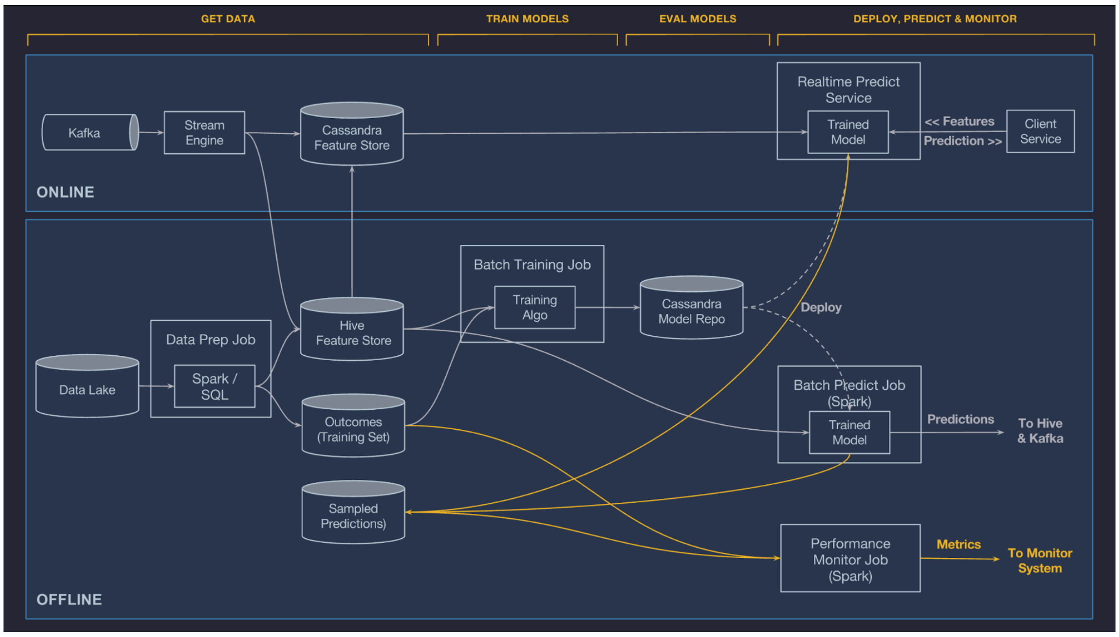


Figure 4.4: Architecture of Uber Michelangelo [11].

Michelangelo is Uber's *Machine Learning-as-a-service* platform, which enables internal teams to easily build, deploy and operate Machine Learning solutions at scale; it is designed to cover the end-to-end workflow: manage data, train, evaluate and deploy models, make predictions and monitor the models themselves. Michelangelo is designed to address the gaps that emerged, by standardizing the workflows and tools. Figure 4.4 shows both the online and the offline architecture of Michelangelo. The platform consists of multiple Open Source tools and different built in-house components; the primary Open Source components are HDFS, Spark, Samza, Cassandra, MLLib, XGBoost and TensorFlow; to manage the resources both YARN and Mesos can be used. Michelangelo is designed to provide scalable, reliable, reproducible easy to use tools to address an end-to-end workflow. Michelangelo is the first architecture that includes a centralized Feature Store; it allows teams to create and manage canonical features to be used and shared.

With respect to the architecture exposed in Chapter 3 Michelangelo is a general-purpose platform that intersects both MLOps and DataOps (cf. Section 1.1.2); it also does not explicitly include an experiment tracking tool, a pipeline orchestrator and a CI/CD tool. Due to these differences, the infrastructure presented in the previous chapter would probably not be suitable for Uber's requirements.

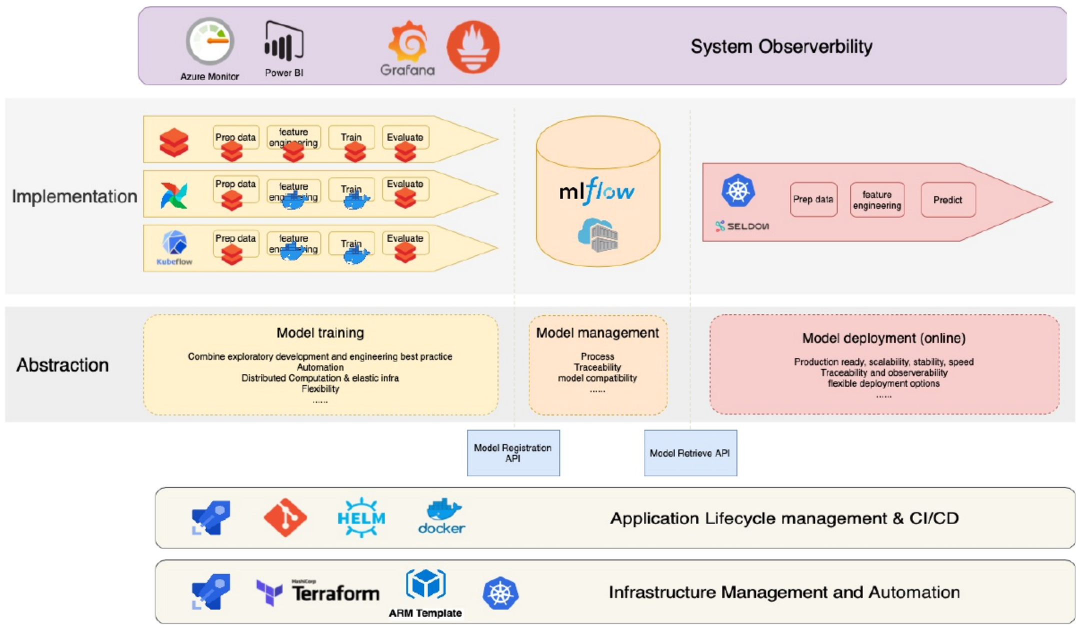

System Observerbility
Azure Monitor
Power BI
Grafana
Implementation
Prep data
feature engineering
Train
Evaluate
Kubeflow
mlflow
SELDON
Predict
Abstraction
Model training
Combine exploratory development and engineering best practice
Automation
Distributed Computation & elastic infra
Flexibility
Model management
Process
Traceability
model compatibility
Model deployment (online)
Production ready, scalability, stability, speed
Traceability and observerability
flexible deployment options
Model Registration API
Model Retrieve API
HELM
docker
Application Lifecycle management & CI/CD
Terraform
ARM Template
Infrastructure Management and Automation

Figure 4.5: MLOps architecture of H&M [12].

H&M "*We have a saying: 'We don't care how good your model is, if it's not in production and delivering value then it's not worth anything'*" [37] says Errol Koolmeister, head of AI foundation at H&M Group. H&M, in its AI department, has many different teams and counts hundreds of models across the entire H&M value chain. Each team solves different business problems, potentially using distinct techniques, however the process is very similar: there is a Machine Learning pipeline, a model deploying pipeline and monitoring infrastructure. Kevan Wang, AI architect in H&M group, underlines the complexity of MLOps and the importance to first think about what kind of problem to address, what kind of process is involved and which kind of skills a team owns [12].

In December 2020 H&M started building its own and centralized AI platform, based on MLOps principles. Figure 4.5 shows the technologies stack for the MLOps architecture of H&M, which is very similar to the reference architecture of this thesis, exposed in Chapter 3; the main difference is that H&M's implementation is a general-purpose architecture, in which the model training pipeline can be handled through Databricks, Airflow or Kubeflow, depending on the specific product lifecycle (instead of Kubeflow only): "*Airflow and Kubeflow are very similar but, if starting from scratch, I'd probably suggest Kubeflow*" [12], says Kevan Wang. CI/CD is implemented, by H&M, by using Azure Native Service as opposed to the reference architecture for this thesis, which adopts Kubeflow and, serving, is provided through Seldon Core instead of BentoML. H&M additionally includes monitoring and system availability, handled through Azure and two Open Source tools: Graphana and Prometheus. Regarding the model management H&M makes the same choice as the architecture in Chapter 3, by considering MLFlow as the most mature solution today on the market.

Chapter 5

Conclusions

MLOps is a very recent approach aimed at simplifying the workflow and reducing the time to get models in production. The goal of this thesis was to provide a deep study of this new methodology. Accordant with the analysis of the features of MLOps, it can be concluded that the Machine Learning workflow can be significantly simplified and, the process to get a model in production, can be accelerated thanks to this approach; this goal can be achieved by adopting techniques as CI/CD, Continuous Training and Monitoring, automation, ensuring repeatability, by designing the Machine Learning system in a modular fashion and by applying all the other features of MLOps explained in Chapter 1. This thesis is the result of the internship at Data Reply and it has also the objective of providing a deep study on the plethora of tools to build an MLOps architecture; the analysis on the different tools was conducted by focusing on the Open Source ones, as requested by the company, in such a way as to be platform-agnostic respect to the cloud platform used to deploy the architecture. Chapter 2 offers an in-depth analysis on the main Open Source tools in the market and on their maturity level. According to this deep exploration of the Open Source tools, it can be concluded that some of them can be considered the best choice in the market and "standard-de-facto" among the MLOps technologies. In addition to a deep study on MLOps' features and a deep analysis on its main technologies, the other main contribution was to implement an MLOps architecture, designed by using some of those tools and deployed on Google Cloud Platform. Due to Google Cloud Platform restrictions Seldon Core, the main serving Open Source tool in the market, cannot be employed; future works concern implementing serving features by Seldon Core itself and including in the reference architecture monitoring tools (for example Grafana).

This thesis can be a starting point to explore MLOps both theoretically and practically (by relying on the implemented reference architecture and its code, published as Free Software with GPLv3 licence). Nevertheless the tools mentioned in Chapter 2 can be considered the best choice in the market, among the MLOps technologies, they are very recent and some of them are still immature. During the implementation of the reference architecture, I ran into different lacuna both using MLFlow and Kubeflow. The full MLFlow API is available only in Python and the Java and R API are still in development; other shortcomings I run into, while using MLFlow, were related to the XGBoost package (which didn't work as expected) and the lack of a method to retrieve the version of a model (information required by other functions within the library). Related to Kubeflow, while using the kfp-cli tool (to dynamically submit and run a pipeline), many Exceptions were not handled and it was very difficult to find the root of the problems. Therefore, even if both MLFlow and Kubeflow are very effective for generic usage, they require an improvement for deep and detailed usage. Furthermore, as mentioned in Chapter 2, a common trait of the exposed tools is a lack in the documentation: this needs to be improved in such a way as to make each tool easier to understand. MLOps is rapidly evolving, maturing and is a topic with great potential in the market: it will grow more and more in the near future. "*MLOps market is expected to expand to nearly US$4 billion by 2025*" [23]. Moreover, MLOps will impact the Machine Learning processes in the same way DevOps had a wide impact on software development in the past. For these reasons many companies are starting to adopting this methodology and it will be charming to have the opportunity to work dealing with MLOps in the future; some use cases of companies, which recently started employing MLOps, are provided in Chapter 4. Based on the study conducted with this thesis, companies dealing with Machine Learning should consider adopting MLOps (but it is crucial doing it according to their specific requirements, as suggested in Chapter 3).

Glossary

A

A/B tests A/B testing is a way to compare the two versions of a variable to find out which performs better in a production environment. 35, 63

C

Concept Drift Concept Drift refers to the phenomenon that happens when the statistical properties of the **target variable** change. 15, 17, 53, 54, 63

containerization Containerization is an increasingly popular solution to deal with dependencies when deploying a Machine Learning Model. 8, 19, 63

D

DAG A DAG is a collection of all the tasks to be run, organized in a way that reflects their relationships and dependencies. ix, x, 23, 30, 31, 40, 41, 47, 49, 63

Data Drift Data Drift is an unexpected and unplanned change in the distribution of the data used in a predictive task. Data drift happens when statistical properties of the **predictors** change. 15, 17, 63

Design Pattern A design pattern is a solution for a problem which occurs over and over again, in such way the solution itself can be adopted many times. 22, 26, 63

Domain Specific Language "DSLs are small languages, focused on a particular aspect of a software system." [44]. 31, 63

E

Ensembled Model Ensemble modeling is a process where multiple diverse base models are used to predict an outcome. 13, 63

F

feature A Feature is a measurable property of a phenomena under observation and (part of) an input to a Machine Learning Model (e.g. a raw word, a pixel, a sound wave, an aggregate, a time window, etc.) . 34, 63

Feature Store A Feature Store is a new layer of abstraction aimed to reduce the time that Data Scientists spend on getting the data into a format they can use to train models and maximize the amount of time they actually do data science. Though it is a repository of different features associated with business entities that are created and stored in a central location for easier reuse. 7, 21, 34, 58, 63, 66

Function as a Service "FaaS (Function-as-a-Service) is a type of cloud-computing service that allows you to execute code in response to events without the complex infrastructure typically associated with building and launching microservices applications. Serverless and Functions-as-a-Service are often conflated with one another but the truth is that FaaS is actually a subset of serverless." [45]. 45, 63

M

metric A number that you care about. May or may not be directly optimized in a machine-learning system. A metric that your system tries to optimize is called an objective. 8, 9, 15, 17, 18, 21, 29, 42, 49, 50, 63

model A Machine Learning model is a mathematical function that relates an input to an output. To do that mapping Machine Learning relies on parameters. It is the representation of what a Machine Learning system has learned from the training data, based on statistical theory. 3, 8, 21, 29, 35, 36, 56, 63

Model Store It is a tool for versioning, exporting, and storing machine learning models that allows to collaboratively manage the full lifecycle. 27, 28, 63

Multi-Armed Bandits Technique (and area of study) to deal with the problem of deciding how to route requests to competing Machine Learning model and determines which model is the best in the shortest amount of time can be treated. 35, 63

O

OpenAPI “The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection.” . 44, 63

P

parameter A parameter is a real valued variable that changes during the model training. A special kind of parameter is an hyper-parameter: it is set before the training and it does not change (until the next epoch or retraining). 15, 17, 21, 27, 29, 30, 32, 42, 49, 63, 66

S

serving The process of taking some sort of trained Machine Learning model and make its predictions available for its users. 28, 29, 35, 36, 56, 57, 60, 63

stale model The model is defined as stale if the trained model does not include up-to-date data and/or does not satisfy the business impact requirements. 18, 63

T

Technical Debt Technical debt is a term related to immature, incomplete or inadequate code (due to design deficiencies, low quality or other problems on the software), which will require additional work to be fixed. It is a metaphor linked to finance: having tecnical debts on a software, in economy terms, would be like paying interest on a loan. Further, technical debt, is related to “deficiencies in internal quality that make it harder than it would ideally be to modify and extend the system [...]” [46]. 3, 12, 13, 63

Bibliography

[1] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden Technical Debt in Machine Learning Systems. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 2503–2511, Cambridge, MA, USA, 2015. MIT Press.

[2] Data Science project lifecycle. `https://coursebricks.com/blog-data-science-project-lifecycle/`.

[3] Survey - Kubeflow continues to move into production. `https://blog.kubeflow.org/kubeflow-continues-to-move-to-production`.

[4] Google. MLOps: Continuous Delivery and automation pipelines in Machine Learning). `https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning`.

[5] Theofilos Kakantousis, Antonios Kouzoupis, Fabio Buso, Gautier Berthou, Jim Dowling, and Seif Haridi. Horizontally Scalable ML pipelines with a Feature Store.

[6] FeatureStore.org. `https://www.featurestore.org/`.

[7] Enrico Salvucci. The reference MLOps architecture. `https://github.com/esalvucci/thesis-mlops-reference-architecture`.

[8] Julie Pitt and Ashish Rastogi. Netflix Presents: A Human Friendly Approach to MLOps — Netflix. `https://www.youtube.com/watch?v=fOSZuONmLbA`.

[9] Itaú Unibanco: How we built a CI/CD Pipeline for machine learning with online training in Kubeflow. `https://cloud.google.com/blog/products/ai-machine-learning/\itau-unibanco-how-we-built-a-cicd-pipeline-for-\machine-learning-with-online-training-in-kubeflow`.

[10] Mlops at GreenSteam: Shipping Machine Learning [Case Study]. https://neptune.ai/blog/mlops-at-greensteam-shipping-machine-learning-case-study.

[11] Meet Michelangelo: Uber's Machine Learning Platform. https://eng.uber.com/michelangelo-machine-learning-platform/.

[12] Keven Wang. Apply MLOps at Scale by H&M. https://databricks.com/session_eu20/apply-mlops-at-scale.

[13] Danilo Sato, Arif Wider, and Christoph Windheuser. Continuous Delivery for Machine Learning. https://martinfowler.com/articles/cd4ml.html.

[14] SREcon19 Asia/Pacific - What Is ML Ops Solutions and Best Practices. https://www.youtube.com/watch?v=ALGxALx46f8&t=156s.

[15] Agile Manifesto. http://agilemanifesto.org.

[16] Continuous Delivery. https://continuousdelivery.com/.

[17] What the Ops are you talking about? https://towardsdatascience.com/what-the-ops-are-you-talking-about-518b1b1a2694.

[18] DataOps Manifesto. https://www.dataopsmanifesto.org/.

[19] M. Treveil, N. Omont, C. Stenac, K. Lefevre, D. Phan, J. Zentici, A. Lavoillotte, M. Miyazaki, and L. Heidmann. *Introducing MLOps.* O'Reilly Media, 2020.

[20] Why Is There No DevOps Manifesto? https://devops.com/no-devops-manifesto/.

[21] Damian A Tamburri. Sustainable MLOps: Trends and Challenges. In *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, page 1. IEEE, 2020.

[22] Ml Model Management and Operations 2020 ("MLOps"). https://www.cognilytica.com/2020/03/03/ml-model-management-and-operations-2020-mlops/.

[23] Infographic: The Rapid Growth of MLOps. https://www.cognilytica.com/2020/04/02/infographic-the-rapid-growth-of-mlops/.

[24] Mlops optimizes development, deployment, and management. https://www2.deloitte.com/us/en/insights/focus/tech-trends/2021/mlops-industrialized-ai.html.

[25] Yue Zhou, Yue Yu, and Bo Ding. Towards MLOps: A Case Study of ML Pipeline Platform. In *2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE)*. IEEE, 2020.

[26] Kaz Sato. What is MLOps? Best Practices for DevOps for ML (cloud next '18). `https://www.youtube.com/watch?v=_jnhXzY1HCw`.

[27] Rules of Machine Learning. `https://developers.google.com/machine-learning/guides/rules-of-ml`.

[28] Jez Humble and David Farley. *Continuous Delivery: reliable software releases through build, test, and deployment automation.* Pearson Education, 2010.

[29] ml ops.org. MLOps principles. `https://ml-ops.org/content/mlops-principles`.

[30] Valliappa Lakshmanan, Sara Robinson, and Michael Munn. *Machine Learning Design Patterns.* "O'Reilly Media, Inc.", 2020.

[31] Setting up an MLOps environment on Google Cloud. `https://cloud.google.com/architecture/setting-up-an-mlops-environment`.

[32] Aws MLOps Framework. `https://aws.amazon.com/solutions/implementations/aws-mlops-framework/`, `https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-projects-why.html`.

[33] Azure MLOps Framework. `https://docs.microsoft.com/en-en/azure/architecture/reference-architectures/ai/mlops-python`, `https://docs.microsoft.com/en-us/azure/machine-learning/concept-model-management-and-deployment`.

[34] Clive Cox, Dan Sun, Ellis Tarn, Animesh Singh, and David Goodwin. Serverless inferencing on Kubernetes. *arXiv preprint arXiv:2007.07366*, 2020.

[35] Sridhar Alla and Suman Kalyan Adari. Introduction to MLFlow. In *Beginning MLOps with MLFlow*, page 125. Springer, 2021.

[36] Kfserving documentation. `https://www.kubeflow.org/docs/components/kfserving/kfserving/`.

[37] Seldon Deploy 1.0 Launch Event. `https://www.seldon.io/seldon-deploy-1-0-launch-event/`.

[38] Sasu Mäkinen et al. Designing an open-source cloud-native MLOps pipeline. *University of Helsinki, Faculty of Science*, 2021.

[39] Forecasting hourly electricity consumption of Germany. https://www.kaggle.com/francoisraucent/forecasting-electricity-consumption-of-germany.

[40] How MLOps helps keep Machine Learning solutions relevant during challenging times. https://medium.com/datasparq-technology/how-mlops-helps-keep-machine-learning-solutions-relevant-during-challenging

[41] Concept Drift and the Impact of COVID-19 on Data Science. https://www.iguazio.com/blog/concept-drift-and-the-impact-of-covid-19-on-data-science/.

[42] Metaflow Doc. https://docs.metaflow.org/introduction/what-is-metaflow.

[43] How These 8 Companies Implement MLOps – In-Depth Guide. https://neptune.ai/blog/how-these-8-companies-implement-mlops.

[44] Martin Fowler. *Domain-specific languages.* Pearson Education, 2010.

[45] What is FaaS (function-as-a-service)? https://www.ibm.com/cloud/learn/faas.

[46] Technical Debt. https://martinfowler.com/bliki/TechnicalDebt.html.