Department of Computer Science
Hochschule für Technik Stuttgart

Hochschule
für Technik
Stuttgart

Master's Thesis in Software Technology

Redesign of Professional Machine Learning Platform Based on an Open-Source and Kubernetes-Native MLOps Pipeline to Reduce the ML-Systems Technical Debt

Konstantinos Loizas

In cooperation with: **Mercedes-Benz AG**

Mercedes-Benz

Author:	Konstantinos Loizas
Supervisor:	Prof. Dr. Marcus Deininger
Co-supervisor:	Mr. Josip Skafar
Submission Date:	February, 2022

Affidavit

I hereby declare that the Master Thesis has been written only by the undersigned and without any assistance from third parties.

Furthermore, I declare that no sources have been used in the preparation of this document, other than those indicated in the document itself.

Stuttgart, February 17, 2022

KONSTANTINOS LOIZAS

Καλό ταξίδι παππού..

Abstract

The development of Machine Learning models has evolved into a vital activity for the smooth operation and the perpetual growth of modern organizations. Nonetheless, one of the most prevalent challenges that enterprises confront during ML operations is the inability to productize their implemented solutions. That phenomenon primarily originates in the technical interdependencies between Data Scientists and Software Engineers. While the first should be responsible only for designing and developing models, often, they need the assistance of the second to deploy, scale and serve their ML code. As a consequence for the organizations, technical debt is generated, leading to time, resource, and eventually monetary costs. To overcome this problem, MLOps, a relatively new concept based on the DevOps method, is utilized by automating the lifecycle of ML systems.

This thesis, implemented in cooperation with the Mercedes-Benz AG, presents the design and the development of a complete MLOps lifecycle, integrated into one of the major company's open-source projects, the Data and Analytics Platform (DnA). The solution is based on existing Free and Open Source Software (FOSS) technologies and tools such as Kubeflow, which were utilized and adjusted to meet the needs of modern organizations. In parallel to the setup process of the MLOps pipeline, research was conducted to gather evidence and data about the enterprise readiness of FOSS solutions. The implemented MLOps workflow eliminates the technical debt and enables Data Scientists to scale and deploy their ML models without requiring technical expertise. Furthermore, a real-life use case scenario within Mercedes-Benz is used to validate the developed solution. Finally, the thesis applies criticism to the security-design nature of open-source software and provides recommendations for the further improvement of the implemented MLOps architecture.

Keywords— MLOps, FOSS, Security, Kubeflow

Acknowledgements

I would like to express my sincere gratitude to my supervisors Prof. Dr. Marcus Deininger and Mr. Josip Skafar, for their consistent support and invaluable advice. Prof. Dr. Deininger has been, during my studies at HFT Stuttgart, a knowledgeable, passionate, and approachable Professor and Software Engineer who fosters critical and scientific thinking. Mr. Skafar, a truthfully open-minded, sympathetic, and experienced engineer, motivated me to think outside the box with his inspirational technological vision.

To my colleague but first and foremost to my new friend Jan Migon, I would like to communicate my most special thanks for two reasons. I would like to thank him for the time we spent together on solving challenges, as this project's implementation wouldn't be possible without his contribution. And secondly, I would like to thank him for the mental and psychological support that only a good friend could offer during the project period: "I never thought I'd die fighting side by side with an elf." "What about side by side with a friend?"

I must also express my profound thankfulness to my family for the continuous encouragement throughout my years of study. This accomplishment is dedicated to my father Hercules, my mother Evangelia, my two brothers Michael and Ioakeim, and my dear grandparents Maria and Michael.

Last but most importantly, nothing would be possible without the unconditional love and support of my dear partner Matina. Thank you for understanding me, patiently supporting me during long and demanding working times, and most appreciably, for always being there for me. I can only pledge in writing that we will make all the trips we postponed during my studies. I am thankful that we share our goals and create our common dreams.

Contents

1 Introduction

This chapter provides a short overview of the company background, a brief description of the thesis task, and finally, a layout of the following chapters.

1.1 Background

This thesis was conducted in cooperation with Mercedes-Benz AG, a Mercedes-Benz Group AG company.

Figure 1.1: The Mercedes-Benz AG logo

1.1.1 Company & Department

Mercedes-Benz Group AG (former Daimler AG), based in Stuttgart, Baden-Württemberg, Germany, is an international group of companies and one of the worldwide leading organizations in the automotive spectrum. The corporation foundation was rooted in 1886 to the invention of the first automobile from Carl Benz and Gottlieb Daimler. Both of them discovered and manufactured the first motor vehicle in the world independently but almost synchronously (Daimler, 2020a). The official establishment of the organization was dated back to 1926 when the two pioneers (Benz & Daimler) merged their companies to cope with the economic crisis after World War I (Haghirian & Kayser, 2018). Today, the Mercedes-Benz Group AG consists of two major organizations: Mercedes-Benz AG (Cars & Vans) and Mercedes-Benz Mobility AG. In addition to being one of the world's largest

manufacturers of vehicles, the company offers financing, insurance, and other mobility services globally while maintaining production units in almost every continent. According to Daimler (2020b), in 2020, the former group numbered around 288,500 employees and sold about 3 million vehicles.

Mercedez-Benz AG uses Agile to enable teams to work efficiently and deliver quality products and services to the customers. Undoubtedly most of the Agile frameworks are predominantly customized for smaller team sizes (Alqudah & Razali, 2016). As a result, larger firms often take advantage of several extended Agile forms to develop and manage extensive projects and teams. The group companies of Mercedez-Benz utilize the Scaled Agile Framework (SAFe) designed by Leffingwell et al. (2018). One of the main characteristics of SAFe is the Agile Release Train (ART). An ART is fundamentally a group of different teams working in cooperation on a shared company value stream (Brenner & Wunder, 2015). Distinct ARTs comprised of people from separate Capabilities (the SAFe term for departments) focus on end-to-end responsibility of various products (Figure 1.2). The Data, Analytics & Functions Enabling ART is one of the core ARTs inside Mercedes-Benz AG. It is responsible for more than five different products related to solutions around business functions, data, and artificial intelligence/machine learning models. The implementation of this thesis took place in the Data, Analytics & Functions ART of Mercedes-Benz AG and, more precisely, it is part of the Data & Analytics platform (hereafter referred to as the DnA platform) development.

1.1.2 DnA Platform

In recent years, more and more organizations worldwide are embracing the use of Free and Open Source Software (FOSS). Mercedes-Benz Group AG acknowledges that FOSS has become a key component in numerous company products and is committed to contributing to the open-source community in various global projects (Daimler, 2020c). Furthermore, most of the group companies are now developing open-source software products. Mercedes-Benz AG launched several FOSS projects to encapsulate the various open-source benefits and give back to the international open-source community.

The DnA platform is one of the first FOSS Mercedes-Benz AG projects. On a com-

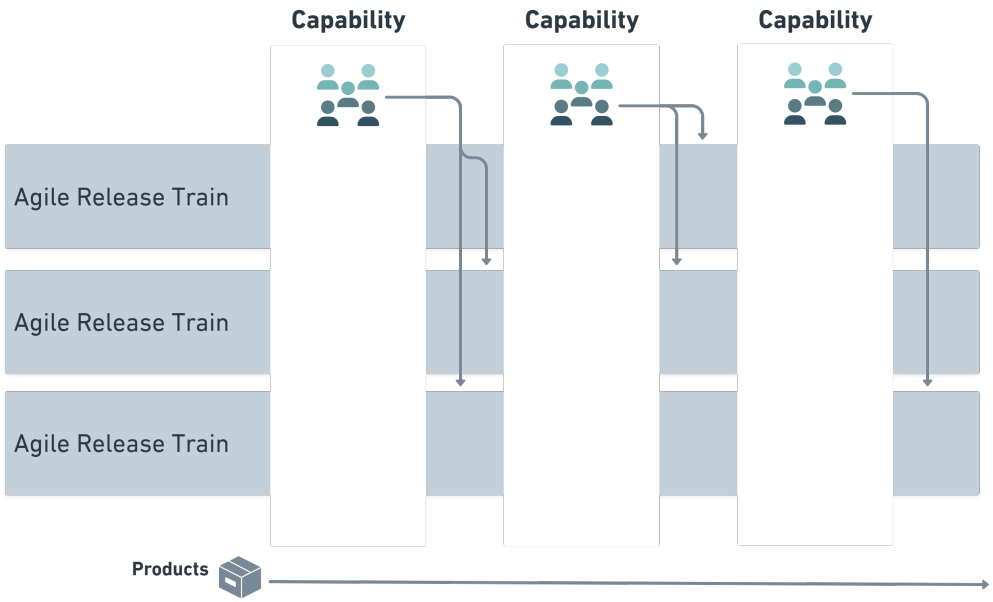

Figure 1.2: SAFe: ARTs, Capabilities & Products

pany level, the main objective of this product is to enable all Data Scientists and non to create Artificial Intelligence or Machine Learning (AI/ML) models in an effective, efficient and compliant way. More specifically, the DnA platform is aiming to formulate a toolkit. A toolkit that will allow anyone to create, manage and share AI/ML solutions without spending additional time on unnecessary configuration steps. That also includes GDPR compliant data access every time used internally. The DnA platform consists of three main parts:

1. The Solution section where users can create, manage, share and access solution descriptions.
2. The Reports section, where users can create and edit reports.
3. The Workspace section, where users can:
    - Create experiment Workspaces, write code using the environment of the open-source Jupyter Notebook, and then create provisions for their solutions.
    - Make use of several services, such as malware scan, etc.

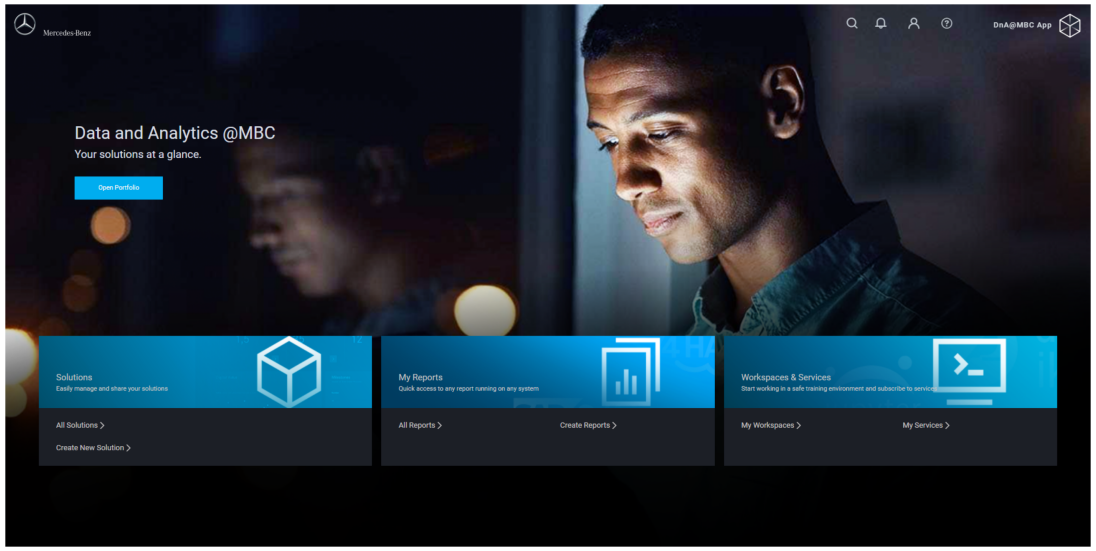

Figure 1.3: DnA Platform: Homepage

1.2 Task Overview

The task of this thesis can be divided into two subtasks. The first subtask outlays the research, design, and development of an MLOps solution in the environment of Mercedes-Benz and the DnA platform. The second includes the leverage of existing Free and Open Source Software for this purpose and the investigation around their applicability to secure enterprise environments.

1.2.1 Problem

The era of big data has brought Machine Learning into a protagonistic position in everyday life. A survey by Algorithmia (Columbus, 2021) found that more than 75% of enterprises gave in 2021 high priority to AI and ML over other IT actions. Nevertheless, the evolution of ML systems is rapid, their establishment large, and their complexity deep. That is the main reason why these systems are accused of being prone to technical debt creation (Tang et al., 2021). The technical debt term was introduced back in 1992 and describes the long-term costs (financial and not) that the blistering evolution of a software engineering framework creates (Sculley et al., 2015a). In the case of Machine Learning systems, technical debt can be produced in both the implementation and maintenance

stages. For instance, costs in ML solutions are often increased when Data Scientists require the involvement of Software Engineers to deploy their models. To tackle these challenges, organizations, and communities globally are now focusing on creating MLOps solutions inspired by the widely used DevOps methods. While several licensed products for MLOps are available, such as Dataiku, there is a need for open-source MLOps alternative solutions. Fortunately, FOSS projects such as Kubeflow or MLFlow exist to help with ML lifecycle management. The possible effects of creating an open-source MLOps lifecycle could dramatically reduce the technical debt. Nevertheless, FOSS is quite often considered not being enterprise-ready.

1.2.2 Objective

The main goal of this thesis is to reduce the technical debt of ML by researching, designing, and implementing an MLOps lifecycle for the Mercedes-Benz DnA platform, using existing open-source tools and technologies. In parallel, of equal importance is the exploration and investigation of how enterprise-ready can the open-source software be. Finally, the ultimate objective is to enhance the Data Scientists' user experience by enabling them to work on the actual implementation of ML models and eliminate their interaction with extra configuration settings by automating them in the background.

1.3 Thesis Outline

The Master Thesis' structure is outlined in the following chapters:

- *Chapter 2* defines the task of this thesis by providing a description of the problem and the reasons that constitute its solution inevitable.
- *Chapter 3* outlays the current state of the art around the problem. In particular, this chapter includes a comprehensive bibliographic overview of the main concepts and frameworks used in modern organizations. Finally, the chapter presents the technologies and the tools used for the solution development.
- *Chapter 4* illustrates the overall approach selected to tackle the problem and in-

cludes a high-level representation of the generated MLOps lifecycle.

- *Chapter 5* contains the implementation of the suggested solution in the enterprise environment of Mercedes-Benz AG. More specifically, the chapter describes all the challenges, the problems related to the setup of the tools, and their respective solutions.
- *Chapter 6* presents the verification of the implemented solution by providing and evaluating the results from its application to a use case.
- *Chapter 7* outlays the conclusions produced by this project, and based on them, provides future suggestions and recommendations.

2 Task Description

According to Meulen and McCal (2018), a significant percentage of AI projects (around 85%) fail to escape from the research environment and reach the production pipeline. That practically is translated to economic, and not only loss for organizations, as the developed models often can not be used in the real-life environment.

2.1 Where does the problem occur?

Several reasons can lead to the non-success of Machine Learning projects. The laboratory development of an ML model is only the first step before an organization can practically utilize the solution. Further steps include the packaging of the application, scaling-out, tuning, instrumenting, and maybe automating the whole process (Figure 2.1). The problem arising is that while writing code and creating models in a lab environment, such as Jupyter Notebooks, is a configuration-wise simple process, everything else is not (Haviv, 2020). Implementing an ML solution in a workspace may require only a couple of Data Scientists, if not one, and the required development time can be counted in weeks. However, that is not the case with the subsequent steps. Data Scientists usually dont have the necessary developer skills to productize their models. Consequently, they often must work for months together with software engineers, not on the actual solution but its configuration. That translates to costs in time, effort, resources, and respectively in money. Finally and interestingly enough, the preceding fact supports one of Boehm (1987)'s statements, already introduced in the 80s, about Software Metrics: *"Software systems and software products each typically cost 3 times as much per instruction to fully develop as does an individual software program. Software system products cost 9 times as much."*

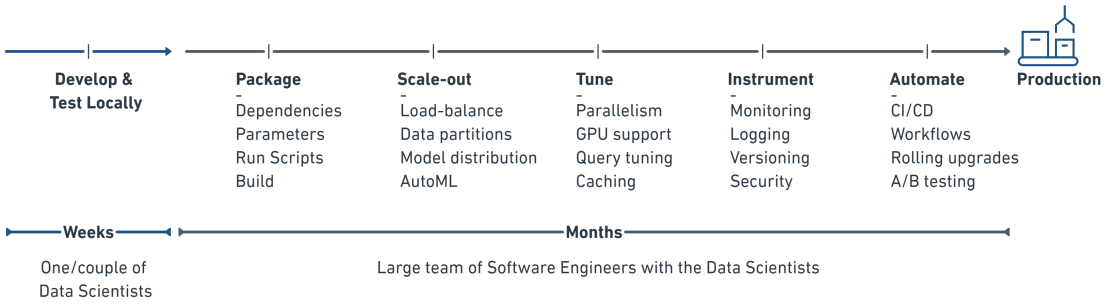

Figure 2.1: ML models from Lab to Production (Haviv, 2020)

In the Mercedes-Benz AG context, that is one of the crucial problems the DnA platform attempts to tackle. More specifically, before the development of the application, the numerous departments of the company around the globe were using custom solutions to develop ML systems. Initially, DnA was implemented to provide centralized and transparent access to data and ML solutions to all the Mercedes-Benz employees. Then, several licensed and open-source development tools and services were added, aiming to constitute the platform as the default choice for all the Data Scientists within the organization. Nevertheless, integrating all the different platform services into an automated lifecycle was the ultimate goal. Since DnA is an open-source project contributed by Mercedes-Benz to the community, the task of this thesis is to create an open-source solution to:

- Reduce the technical debt of ML systems,
- Create automated ML pipelines
- Enhance the Data Scientist experience
- Remove the silos between models and production

2.2 A research question

A critical part of this thesis is to provide a solution utilizing existing open-source technologies and tools. While the modern trends in the IT World (Driver & Klinect, 2019) seem to place open-source software as a defacto standard in more than 95% of the IT enterprises, a distinctly interesting research question can be extracted by this thesis:

- *Is existing open-source software enterprise-ready?*

In other words, as the task requires the utilization of existing open-source solutions, a fundamental aspect of this research is to investigate the behavior of the tools in a substantial and paradigm enterprise environment such as the Mercedes-Benz one. The term behavior is used to describe the collation of the installation process between a local security-free and an enterprise security strict environment. The goal is to identify the existence or not of the security design in the open-source tools and list the required modifications. Based on this exploration, this thesis will conclude how open-source solutions should be designed to be utilized by organizations in real-life cases.

3 State of the Art

This chapter presents a comprehensive literature overview of the most current knowledge around methodologies and tools relevant to the subject of the thesis. More specifically, studying and understanding the following concepts is considered a prerequisite for perceiving the different dimensions of the problem and the possible approaches to its solution. In the first sections, the main goal is to introduce the updated status around significant theoretical ideas and practical methodologies. That will help in formulating the context of the task and composing feasible proposals to tackle it. Finally, the last section presents a thorough outline of the technologies and the tools that can help implement the solution.

3.1 FOSS

According to Stallman (2009), the FOSS term refers to Free and Open Source. During the 1960s and 1970s, free distribution of the programming code produced by researchers in either academic or corporate environments was an unwritten rule of the scientific philosophy (Andersen-Gott et al., 2012). In the 1980s, MIT sold some code developed by its researchers to a private enterprise. This event was responsible for the genesis of the FOSS movement. During that period, Richard Stallman, a researcher at MIT, commenced the Free Software Foundation. The main objective of that organization was to express its opposition against the commercial exploitation of software and promote the core ideas of FOSS (FSF, 2021). The ideology behind open-source software is that developers are freely authorized to operate, customize or share the source code of an application, as long as they are adhering to specific copyright limitations (Ebert, 2008). Access to FOSS automatically guarantees code enhancement, more efficient bug discovery, effective er-

ror correction, and finally, software optimization to distinct requirements and hardware systems (Bonaccorsi & Rossi, 2003). However, it is crucial to clarify that free software doesn't mean gratis. Stallman has numerous times described it as "*..a matter of liberty and not price · To understand the concept, you should think of free as in free speech, not as in free beer.*" (Stallman, 2015).

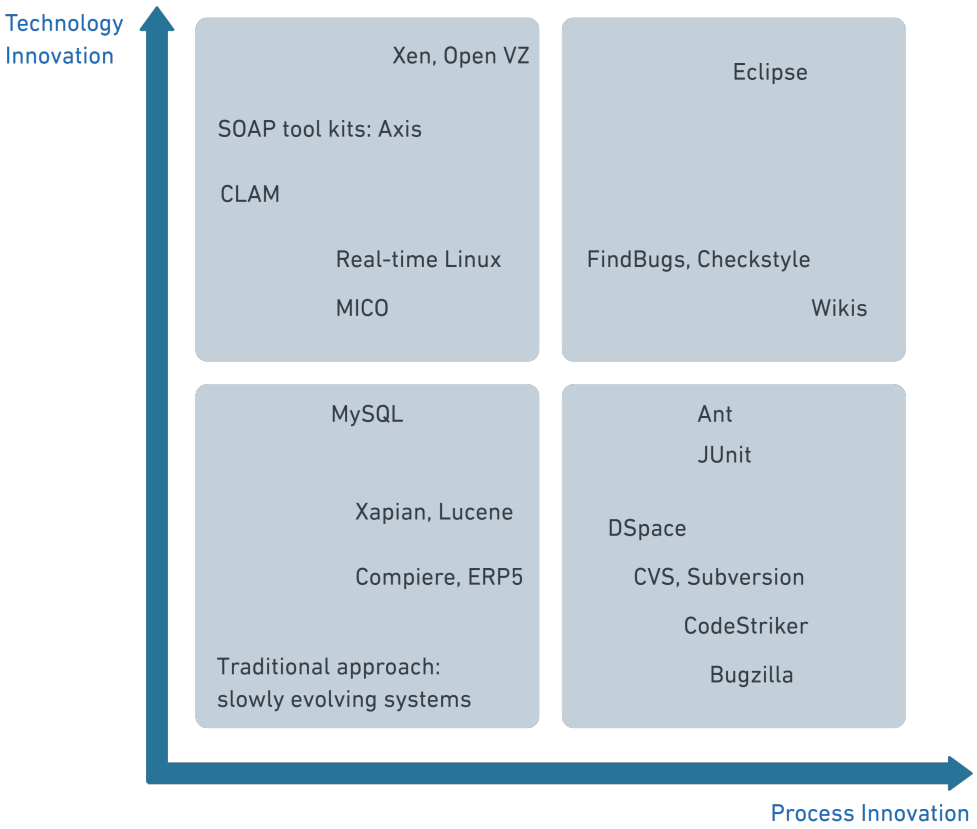

Figure 3.1: Innovative FOSS solutions (Ebert, 2009)

During the last years, more and more global organizations such as Google, Microsoft, and IBM, are investing in FOSS (Gürsakal et al., 2021). Nowadays, conventional software development, in which an organization creates and produces a service or product from the ground up, is rare because software architectures have evolved, they are more complex, and companies are mainly interested in developing their applications using existing frameworks (Ebert, 2008). The reasons that gradually lead more and more corporations to open-source software can be justified by several arguments. Bonaccorsi and Rossi (2006) distinguish the benefits companies can get by participating in open-source activities in three major categories: economic, technological, and social. For the first category,

several studies (Dahlander & Magnusson, 2008; Dahlander, 2005; Ågerfalk & Fitzgerald, 2008; Ebert, 2009) indicate that using standard FOSS components creates profit for the companies as they can focus on the core of the product they design. In parallel, the maintenance, bug spotting, and future improvement costs of open source applications get reduced since the community developers are the ones who often do this job by code contributions (Andersen-Gott et al., 2012). Besides, open-source software with support by large communities tends to be more of high quality because the code is assessed constantly by more developers than a company's department (Ebert, 2009). Next, from the technology perspective, FOSS triggers innovation (Figure 3.1) because, when source code is freely accessible to everyone, radical ideas can be converted to new systems almost straightforwardly (Ebert, 2007). In addition, according to Andersen-Gott et al. (2012), several international firms choose to transform their innovation strategies to "open" because they recognize that they don't have enough resources to discover or hire every single genius developer. Finally, the social motivation of companies to open source communities is an unwritten rule, a norm, that is accomplished by either contributing to existing FOSS projects or creating new open-source services. In case of a violation of this rule, corporations may have to suffer severe consequences since the trust of the community contributors is affected (Bonaccorsi & Rossi, 2006). To conclude, companies that use and contribute to FOSS can gain competitive advantages, as repeatedly proven through literature and real-world examples. Mercedes-Benz AG fosters the utilization and development of open-source systems, such as the DnA platform, to create a mutual benefit among the company and the open-source communities.

3.2 Machine Learning

In the modern world, massive amounts of data are constantly being generated, whereas experts predict an even bigger explosion of the data quantities in the near future (Miller, 2022). Furthermore, the existence of that many data creates an undeniable need for their analysis, the extraction of useful information, and finally, the creation of practical applications based on them (Angra & Ahuja, 2017). Machine Learning (ML), a subfield of Artificial Intelligence (Shinde & Shah, 2018; Chauhan & Singh, 2018), is one of the most

effective methods to process data and create predictions utilizing different models. It is an advanced part of computational algorithms developed to acquire knowledge from the encompassing environment, mimicking human intelligence (El Naqa & Murphy, 2015). In ML, a computer program executes various tasks, and it is supposed that the machine has learned from its experience if its measured performance in these tasks improves as it obtains more and more knowledge (Ray, 2019). According to Mitchell (1997): "*A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E"*. There are four major categories of ML algorithms that one may select to use, depending on the type of data that are available for training (Ray, 2019):

1. *Supervised learning*: The desired solutions, referred to as labels, are included in the training data fed to the ML algorithms (Kang & Jameson, 2018). Supervised learning creates a process where the predicted results are compared to the actual results of input data (known as "training data") and constantly updates the model until the results match the accuracy expected (Patel, 2018). Typical algorithms are Support-vector machines, Linear regression, Decision trees, Neural networks, K-nearest neighbor algorithm, etc.
2. *Unsupervised learning*: The training data don't have labels (Kang & Jameson, 2018). Algorithms like clustering are used to surmise the inherent data connections (Patel, 2018). Typical unsupervised learning algorithms are K-Means, etc.
3. *Semi-supervised learning*: Training datasets include mostly untagged data, but some labeled data are there as well (Kang & Jameson, 2018). Semi-supervised learning can be characterized as a supervised learning adjunction (Patel, 2018). Typical semi-supervised learning algorithms are Laplacian support vector machines, etc.
4. *Reinforcement learning*: The algorithm learns on its own a "policy" (another term for "the best strategy") about how to act in a given circumstance to obtain the maximum rewards (Kang & Jameson, 2018). Compared to supervised learning, it does not expect correct input/output data sets (Patel, 2018). Typical reinforcement learning algorithms are Monte Carlo, Q-learning, etc.

Over the last decades, Machine Learning has been widely adopted by several applications in various areas, mainly due to the evolution of computational power (Boutaba et al., 2018). Robotics, natural language processing, search engines, video games, crime prediction, and social media are only some of the many domains where Machine Learning is used. In addition, the rapid development of cloud computing solutions enhances the application of ML, as hardware such as GPUs and TPUs enable faster training of large amounts of data (Boutaba et al., 2018). To summarise, Machine Learning plays a crucial role in offering solutions for real-life problems by extracting knowledge from a large quantity of accessible data (Alzubi et al., 2018). There are different ML algorithms available to be utilized by organizations and researchers to produce safe and rational decisions based on available data.

3.3 Data Engineering

The rapid increase of available data, in combination with the standardization of the Machine Learning field in most modern organizations, has triggered an expansion of the specialized roles inside the data teams (Saltz et al., 2016). In particular, the vast and infinite data production has created the need for an area that will help in the data preparation for Data Scientists. That area is known as Data Engineering. In fact, this is not more than a further evolution of a widely used and known field that involves database technology and tools: Data Preprocessing (Klettke & Störl, 2021). In his book, Kretz (2019) describes Data Engineers as the connecting interface between an organization's Data Strategy and the Data Scientists working with the data. Undeniably the Data Engineering subject is closer to Software Engineering than to Data Science. In the first two fields, engineers are working on developing software to be used by others to generate results (Saltz et al., 2016). As aptly described by Reddi et al. (2021), Data Engineers are responsible for designing, implementing, and managing data creation pipelines for Data Scientists, using techniques that result in time and cost-efficient solutions.

While the history of the database management systems is rooted in the past, the technological evolution has brought several changes towards different dimensions. In

a recent study, Klettke and Störl (2021) comprehensively distinguish and describe the different generations of the Data Engineering methods:

1. *Data Preprocessing* is one of the most time-consuming and complicated tasks in data science solutions. Therefore, it was progressed to a discrete field: Data Engineering. It includes several suboperations such as Data Understanding and Profiling, Cleaning and Data Correction, and Data Transformation.
2. *Data Engineering Pipelines* are the combination of data algorithms in repeatable executed pipelines. There are several toolboxes available and applicable to different data formats.
3. *Data Engineering Workflows* are related to the selection of the most suitable algorithms for a specific task and their combination.
4. *Automatic Data Curation* is one of the most popular approaches to the solution process automation of the data engineering tasks. It aims to either automate the data curation process and execution of subtasks or generate recommendations on which experts can base their decisions.

To conclude, Data Engineering is actually not a new field but the needed evolution of existing tools and methodologies to support the rising demand for data processing. The number of available data is constantly increasing, and thus the need for processing them is continuously getting more significant. Data Engineering is crucial in this operation, as Data Scientists should only focus on developing the models to exploit the data and not on preparing them for this operation.

3.4 DevOps

In today's advanced technological environment, where cloud applications are predominant, software delivery and updates for customers are expected to take place in a continuous, fast, and efficient way (Lwakatare et al., 2016). Furthermore, to achieve the above-noted, modern organizations strive to invent or implement new approaches to software development, different from the traditional methods. Plenty of predicated agile

methodologies already enable the engineering teams to rapidly adapt to the continuously changing requirements, restrictions, or customer demands during the lifecycle of a project while maintaining and enhancing it at the same time (Cois et al., 2014). Originated in the agile movement (Leite et al., 2020), DevOps comprise a collection of practices that empower the collaboration and communication between developers and contribute to quick, reliable, and quality software delivery (Perera et al., 2017). On the contrary of the considerable acceptance and implementation of DevOps from several organizations around the world, according to the bibliography (Senapathi et al., 2018; Lwakatare et al., 2016; Jabbari et al., 2016), there is still no standard definition of the term. However, almost every literature research (Jabbari et al., 2016) agrees that the formation of the term DevOps is the combined result of the words: Developers and Operations. The main objective of this software paradigm is to eliminate and eventually overcome the organizational silos (Lwakatare et al., 2016) by enabling cross-functional cooperation and trust between the stakeholders of software development activities. Consequently, and according to Perera et al. (2017), DevOps enhances the continuous development (CD) target of the agile methodologies with the continuous integration (CI). Hence it enables fast customer serving and effective market competition for companies.

Organizations can make use of DevOps practices to speed up innovation. That essentially includes the creation and automation of software development and infrastructure management pipelines (Freeman, 2019). Even though in literature can be found many approaches around the practices, this thesis focuses on the six most popular operations:

1. *Communication and Collaboration* enhance the cooperation between developers and operations (Jabbari et al., 2016). In general, organizations utilize chat apps, project tracking systems, and wikis to establish strong culture rules around information sharing (Freeman, 2019). Therefore, this allows all company sections (even other departments like marketing and sales) to align more closely on goals and projects by speeding up communication. Collaboration practices frequently empower team members, particularly developers, who gain more influence over system operability (Lwakatare et al., 2016).

2. *Continuous Integration* enables software engineers to integrate, into a central repos-

itory, all the changes they make in code frequently, followed by automated builds and tests execution. As a result, the bug discovery is faster, the quality of the software is improved, and the verification/distribution time of software updates is shorter (Freeman, 2019).

3. *Continuous Delivery* assures that the code is automatically built, tested, and configured for a production release every time there are changes. After the building stage, CD extends the concept of CI by deploying changes to either a test or production or both environments. CD guarantees that the developers will always access an artifact that has passed some standardized tests (Freeman, 2019).

4. *Microservices* is a cloud-native architecture where a single software application is built as a bundle from small independent services and each of which: a) has a unique scope, b) is autonomously deployable, and c) runs its process (Balalaie et al., 2016). Typically, the different microservices communicate using application programming interfaces (APIs) (Freeman, 2019). Hence, microservices, among other things, enable scalability, easier bug detection, and faster shipping times.

5. *Infrastructure as Code* outlines the utilization of code for infrastructure management and the application of software development techniques, like version control or continuous integration (Lwakatare et al., 2016). With IaC, the interaction between infrastructure and engineers is accomplished with code-based tools, allocation of application environments is faster, and the deployment of the application at any scale can be done automatically (Freeman, 2019).

6. *Monitoring and Logging* is a crucial practice that includes the implementation of a continual feedback loop from the development to the production environment (Lwakatare et al., 2016). As the service's availability must be constant and remain uninterrupted, the data and logs created by applications and infrastructure can be collected, classified, and then examined to quickly identify the root cause of errors or unexpected behavior (Freeman, 2019).

In conclusion, DevOps has become a de facto set of software development practices. And not without reason, as it is already multiple times proven that it can benefit organizations with fast innovation, speedy delivery, reliability, and more when applied

effectively.

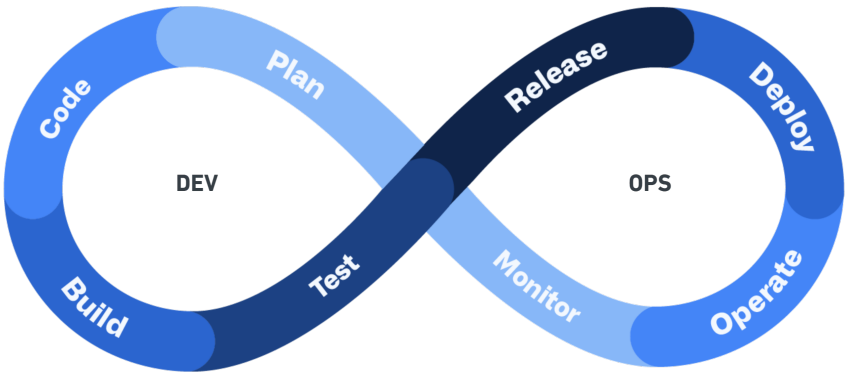

Figure 3.2: The DevOps loop (Cekic, 2021)

3.5 MLOps

MLOps (Machine Learning Operations) is an adequately new term related to machine learning models and software development. However, it has already managed (Figure 3.3) to become a rapidly increasing trend in Google searches. This fact indicates the growing attention to MLOps by both the scientific and corporate environments (Tamburri, 2020). Essentially, MLOps identify as DevOps for ML activities. Their underlying difference from DevOps relies on the characteristic that separates the ML models from the traditional software: data (Breuel, 2020). Consequently, MLOps, are considered a combination of three parts: DevOps, Machine Learning, and Data Engineering (Figure 3.4) (Zhao, 2020).

The importance of DevOps principles for ML workflows is crucial since the development of machine learning models has taken a radical position in many organizations (Karamitsos et al., 2020). The core work of a Data Scientist is very often reputed to be mostly around operations such as the development, training, and evaluation of ML models. Nonetheless, as Sculley et al. (2015b) indicate in their research, the model's code usually comprises only a fragment of the total operations around an ML system (Figure 3.5). An ML workflow, basically a pipeline, includes several steps (Figure 3.6) that are executed typically in different infrastructures. Furthermore, it is often normal that Data

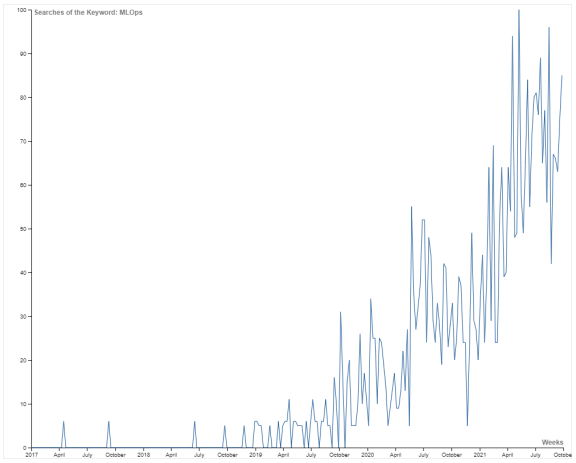

Figure 3.3: Google (2021a) Trends for MLOps searches for years 2017-2021.

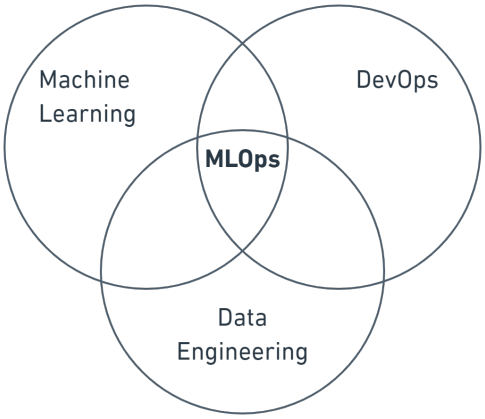

Figure 3.4: MLOps

Scientists don't have the special engineering skills required to configure the various execution environments. In his paper, Tamburri (2020) mentions that about 75% of Data Scientists are not Computer Scientists. Hence, this can lead to technical debt for an organization (Sculley et al., 2015b), including potential errors or overuse of resources (i.e., need for support by Software Engineers). MLOps aims to bridge this gap equivalently to how DevOps assists in the fast development, testing, and deployment of less error-prone and more quality software (Soh & Singh, 2020). The application of MLOps secures the automation and monitoring of all the steps involved in an ML pipeline, such as integration, testing, releasing, deployment, and infrastructure management (Google, 2020). By automating all of the stages needed in building a machine learning system, from development to deployment, MLOps reduces the technical dept and creates reliable and efficient ML systems (Ruf et al., 2021).

3.6 Tools & Technologies

It should be noted that the development of the solution in this thesis is based on various technologies and tools. Nevertheless, the existence of Jupyter Notebooks in the DnA platform constituted the starting point and the basis of the project. Following is presented a comprehensive overview of the most essential tools that were used throughout the research and implementation phase.

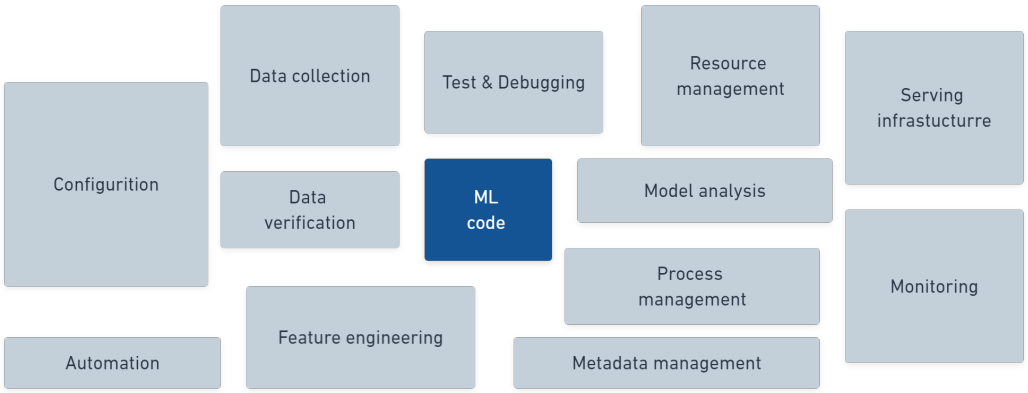

Figure 3.5: Typical elements of an ML system (Sculley et al., 2015b)

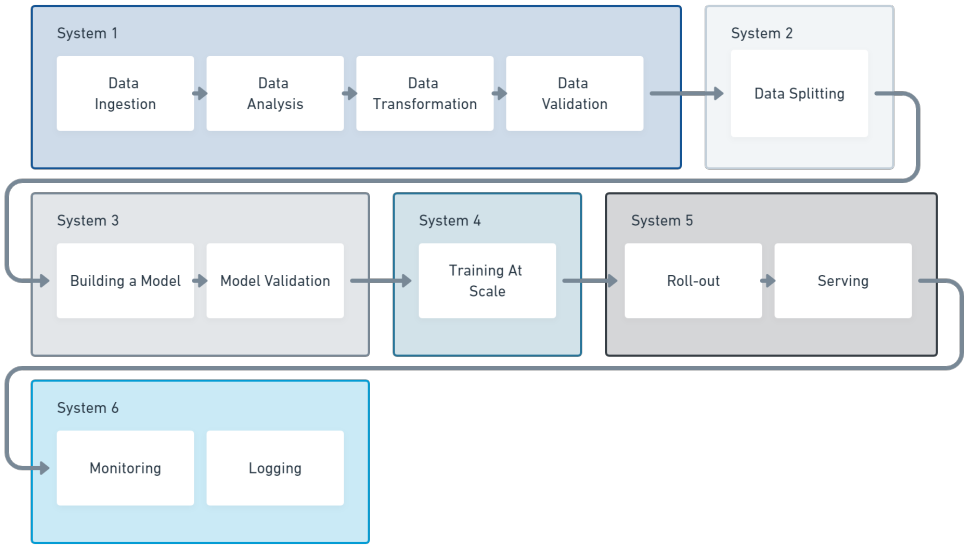

Figure 3.6: A typical ML pipeline (Google, 2021b)

3.6.1 Jupyter Notebook

The Jupyter Notebooks is an open-source computational notebook accessed through a web browser. More specifically, it is a tool where users can combine code development, data, visualizations, explanatory text, and equations in a single document, acting as a virtual lab notebook (Randles et al., 2017). Furthermore, Jupyter Notebook supports programming in multiple languages (K, 2020). This characteristic boosted the popularity of the tool among researchers significantly. Indeed, according to a GitHub study in 2018, around 2.5 million Jupyter notebooks were published on the platform, 200,000 more than in 2015 (Perkel, 2018). In addition, unlike other IDEs like VSCode, Jupyter Notebook is

Figure 3.7: The tools used for the development of this thesis.

pretty handy when it comes to exploratory data analysis (EDA) because it provides an inline preview of the code results independently from other parts (Das, 2021). Nowadays, it is considered the standard tool for data scientists and the development of end-to-end data science workflows because it offers an interactive way to write code and easily combine it with explanatory text or multimedia (Perkel, 2018).

3.6.2 Kubeflow & Kubeflow Pipelines

Kubeflow is an open-source suite of ML tools for Kubernetes originated from an internal Google project. Its goal is to make the deployment of ML systems on Kubernetes faster and easier to manage by containerizing the components of the pipeline (Figure 3.6) and placing them on the cluster in an abstract of technical difficulties way (Bisong, 2019). More specifically, Kubeflow is a Kubernetes-native platform with several components (Figure 3.8) responsible for orchestrating, deploying, and executing scalable ML workloads (Patterson et al., 2021). These components can be used holistically via installing the entire Kubeflow suite, but at the same time, many of them can be utilized for specific uses cases as standalone applications.

Kubeflow Pipelines is one of the core Kubeflow tools. It is responsible for building and managing end-to-end ML workflows, based on Docker containers (Kubeflow, 2021b), on Kubernetes infrastructure. As presented in Section 3.5, the development of a machine learning system includes a set of several tasks which typically can not be represented by a single script. In addition, regularly, many parts have to be changed as the model is being developed. Kubeflow Pipelines combine all these different tasks modularly by creating a form of a directed acrylic graph (Vasconcelos, 2020). Essentially, a pipeline entails the required input parameters of each pipeline component and the in-

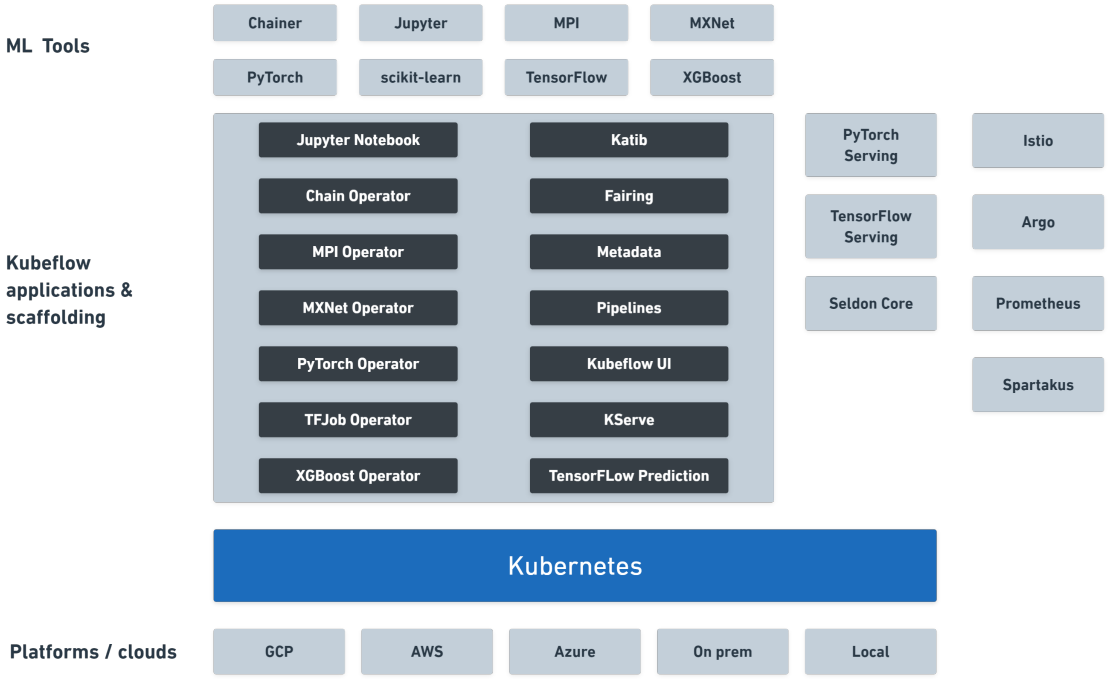

Figure 3.8: The components of Kubeflow (Kubeflow, 2021a)

puts and outputs of each of them. Every pipeline component is a Docker image package of a self-contained code set (Smedinga & Biehl, 2020). Since Kubeflow is a Kubernetes-native platform, during the execution of a pipeline, a single or multiple pods are spawned by the system to start the Docker containers, which accordingly execute the code sets. Thanks to the containerized architecture, Kubeflow Pipelines enable a simple and easy way to reuse, exchange or even replace different parts of the ML workflow at any time (Patterson et al., 2021). The main components of the tool include (Bisong, 2019):

1. A user interface (UI) to manage and track machine learning pipeline runs, experiments, jobs and enable easy collaboration between Data Scientists.
2. A scheduling engine for multi-step machine learning processes.
3. An SDK in Python -particularly handy for Data Scientists- to create or modify the pipelines and their components.

Kubeflow Pipelines can act as a tool to leverage the gap between Kubernetes and MLOps. But despite its distinct and obvious benefits still, there is plenty of specialized work required to create a pipeline. More specifically, the typical steps include (Bisong, 2019):

1. Writing the ML code
2. Creating the Docker images
3. Writing some form of DSL code for Kubeflow Pipelines
4. Compile the DSL code
5. Upload the pipeline to Kubeflow Pipelines
6. Run the Pipeline

Undoubtedly this is not productive as in case of possible changes -which tend to happen very often- one must start again from the second step. Next is presented Kale, a tool responsible for optimizing the workflow of the pipeline creation.

3.6.3 Kale

Kubeflow Automated PipeLines Engine (Kale) is an open-source tool designed to simplify the conversion of ML models written in Jupyter Notebook into Kubeflow Pipelines. In particular, Kale is an add-on for Jupyter Hub that significantly reduces the boilerplate steps 4 required to deploy a Kubeflow Pipeline by providing a click-button UI (Guerrero, 2021). The core idea of Kale is the generation of a Python script by exploiting the JSON structure of Notebooks (Fioravanzo, 2019). That is realized by annotating both the Notebook (Notebook metadata) and its different cells (Cell Metadata) in order to assign them to the appropriate pipeline components and declare their interdependencies (Frikha, 2021). Then, the python script is executed to convert the model into a Kubeflow Pipeline, run the pipeline and save it. Kale consists of four main modules which assure its functionality (Fioravanzo, 2019):

1. The *nbparser* takes the Jupyter Notebook as input, parses it into metadata information, and creates an internal graph representation of how the pipeline will eventually look like.
2. The *static analyzer* is used for the identification of the dependencies between the pipeline blocks.
3. The *marshal* module injects the data between the different pipeline steps.

4. The *codegen* is responsible for generating an executable Python script that spawns and deploys a pipeline.

Moreover, Kale provides the opportunity to execute individual pipeline steps on GPUs, which can dramatically reduce the costs of running a pipeline and, at the same time, secure the power required for the computationally intensive parts. Last but not least, the tool supports the usage of Kubeflow Katib, a component for hyperparameter tuning and neural architecture search. In conclusion, Kale notably enhances the Kubeflow Pipelines role as a Kubernetes MLOps tool since it simplifies the formation of a pipeline directly from the model's source code. With the direct conversion of a Jupyter Notebook into a KFP pipeline, Kale secures that the processing building blocks are appropriately classified and independent from each other (Fioravanzo, 2019). The number of the steps described in 3.5 is remarkably reduced, and at the same time, editing a pipeline becomes easy. Therefore, Data Scientists can save up to three times more time for developing ML models (Figure 3.9).

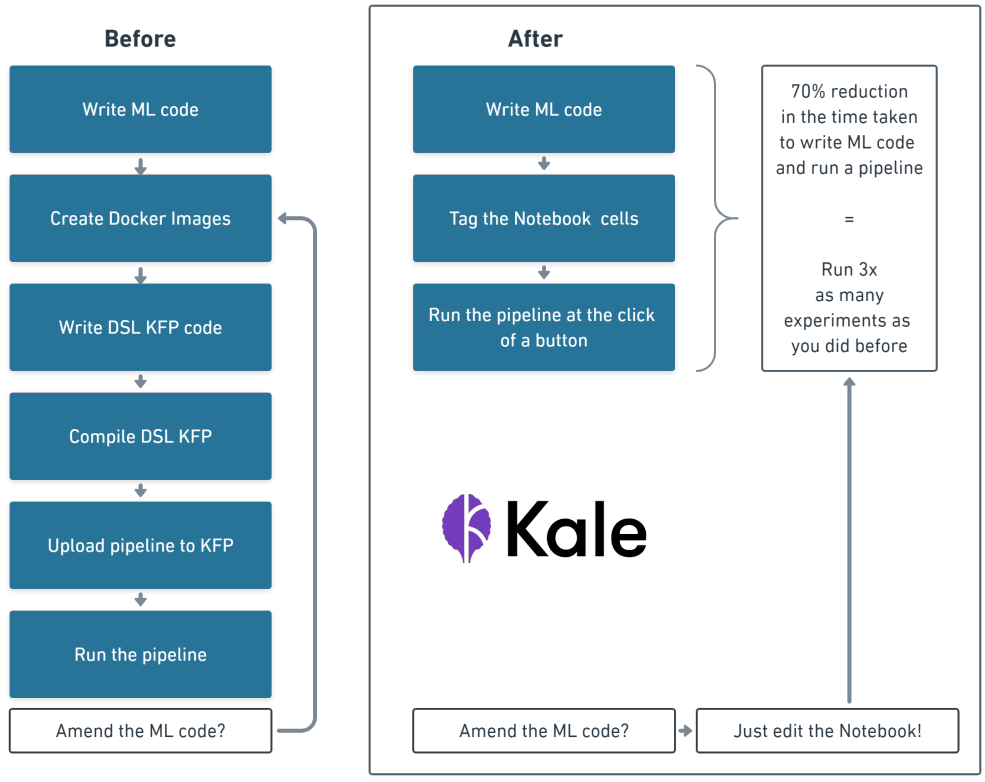

Figure 3.9: Kale reduces the steps for the creation of a Kubeflow Pipeline (Fioravanzo & Koukis, 2020)

3.6.4 KServe

KServe, formerly known as KFServing (KServe, 2021), is a FOSS cloud-native tool specialized in machine learning models serving by providing a Kubernetes Custom Resource Definition. That is an object that extends the Kubernetes API (Vasconcelos, 2021). KServe supports several model-serving systems like TensorFlow, PyTorch, Nvidia Triton Inference Server, etc. Its main goal is to constitute the standard way of serving models, deploying and monitoring inference services, and ultimately reduce significantly the time required by the data scientists to put their models in production (Patterson et al., 2021). It is adopted by major organizations like Nvidia, IBM, and Cisco. The main components of the tool (Figure 3.10) include two widely used cloud-native technologies (Vasconcelos, 2021):

1. *Knative* is used for deploying and managing serverless workloads, something which ensures auto scaling and thus optimization of the costs based on the demand.
2. *Istio* is used as a service mesh technology resulting in imperative features such as Canary roll outs, load balancing, security, and more.

Thanks to these core components, KServe can run on a compute cluster that includes a variety of hardware (GPU, TPU, CPU). Furthermore, the most crucial element of KServe is the Inference Service. It is responsible for managing the served ml models' life cycle, and it is the one to call when an inference from a hosted model is required (Patterson et al., 2021). The advantage of an Inference Service is that serving a model on KServe can happen either via a cli using a YAML file or via the Python KServe SDK (Patterson et al., 2021). As a result, KServe can be used as the standard framework for serving models in production from both MLOPs Engineers who tend to prefer command-line tools and Data Scientists who usually select Python code, bridging the gap between them.

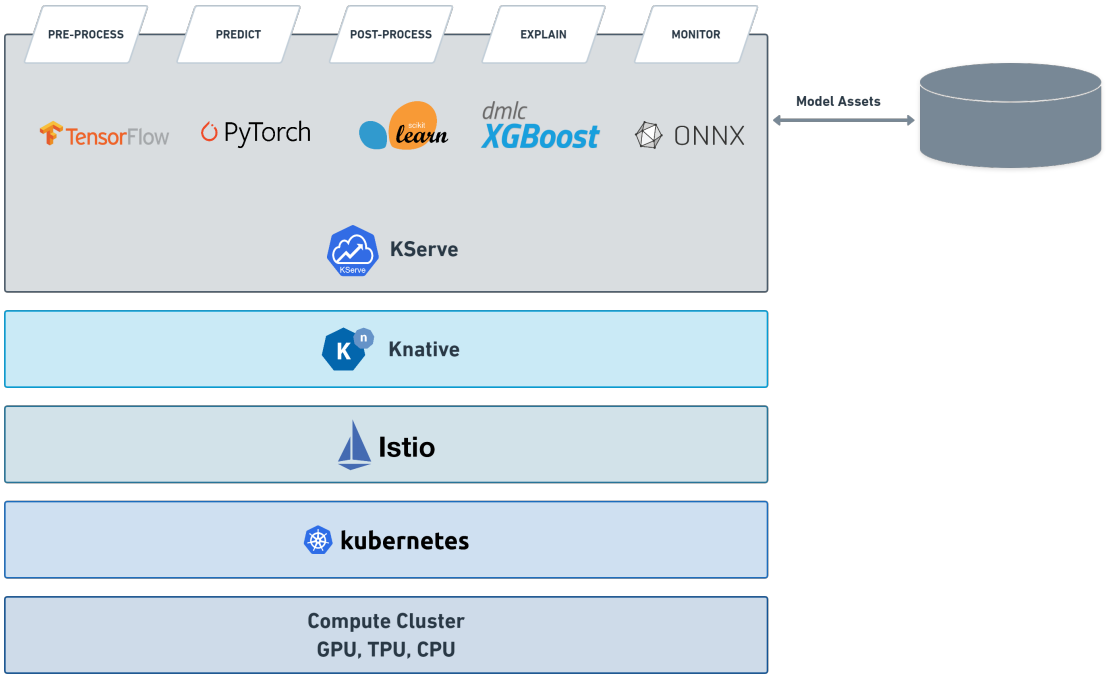

Figure 3.10: The architecture of KServe (Patterson et al., 2021)

4 Overall Approach

The outcome of the theoretical research around the Frameworks, the Technologies, and the tools described in Chapter 3 was the composition of an MLOPs workflow for the DnA platform. This chapter presents an overview of the CI/CD pipeline designed to automate and holistically monitor the development phases behind ML systems.

4.1 MLOps Workflow

The design of MLOps lifecycles in the modern environments where ML models are being developed is a requirement for the age of AI. The reasons, which already analyzed in Chapter 3, are multiple but mainly focused on two problems: the number of the ml systems that fail to reach production is specifically high, and the technical debt between Data Scientists and Software engineers leads to time and resources costs. Google (2020) identifies that the automation of an ML CI/CD pipeline includes six distinct steps (Figure 4.1):

1. Development and experimentation include the development of the ML solutions and outputs to the source code of the models.
2. Pipeline Continuous Integration (CI) describes the build process of the source code and outputs to pipeline components.
3. Pipeline Continues Delivery (CD) outlays the deployment of the CI outputs.
4. Automated Triggering is related to the automatic Pipelines execution due to a scheduler or a trigger.
5. Model Continuous Delivery is achieved by serving the model and making it avail-

able as a prediction service.

6. Monitoring refers to the information collected about the model based on the predictions and constitutes the set-off to start a new model cycle.

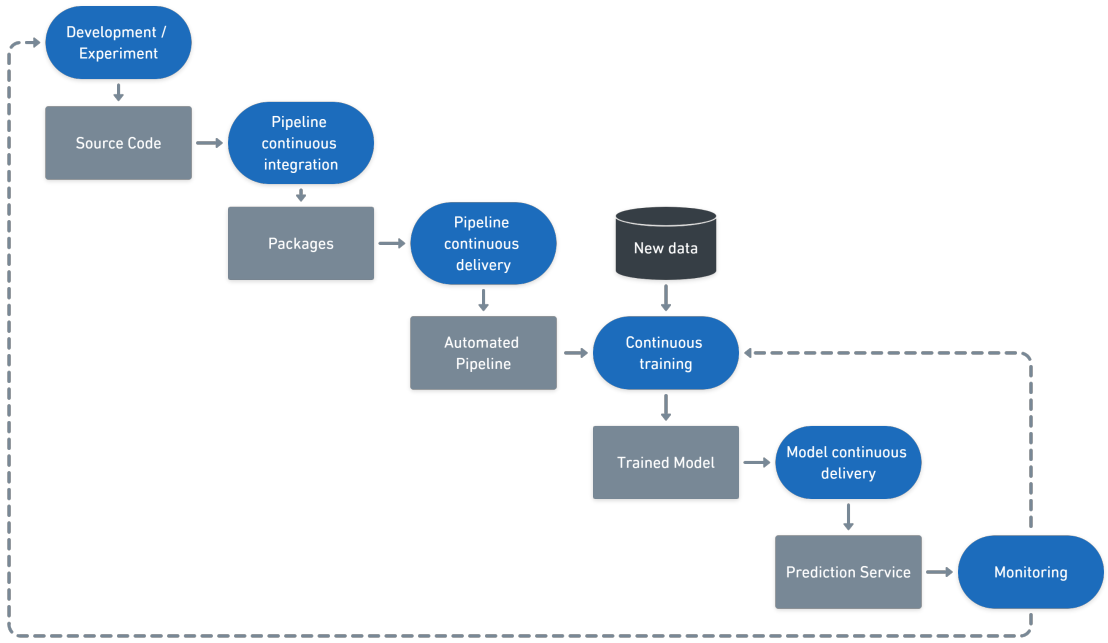

Figure 4.1: The 5 stages for MLOps by Google (2020)

Figure 4.2 presents the diagram of the MLOps workflow designed for the DnA Platform. The basic layout of the lifecycle was based on a combination of the open-source services the platform was initially offering (Jupyter Notebooks with Git integration) and the selected open-source tools that originated in the Kubeflow ecosystem. The pipeline fulfills the six stages described by Google. The users of the DnA Platform can utilize Jupyter Notebooks to develop their ML models while storing the different code versions in GitHub. When their source code is complete, they can use Kale directly from Notebooks to:

1. Build and deploy with a click of a button, portable, scalable, and containerized ML workflows of their models in Kubeflow Pipelines
2. Store their ML models to a registry and serve them via KServe

The Monitoring stage is achieved by getting predictions using simple POST requests to the models' APIs. That can be done either directly from Notebooks or via any CLI.

Furthermore, everything is reachable via the handy and user-friendly UI of Kubeflow Central Dashboard. There, users can access their deployed Pipelines and served Models, organize experiments, share their solutions to multiple contributors, schedule automatic Pipeline executions, get insights about artifacts from their systems, and much more.

However, installing and integrating all these tools and services was rather complicated and effortful. The next chapter presents all the challenges encountered during the setup of the designed ML workflow and their respective solutions. The outcome of this process is additionally providing data and information, interesting for the research question presented in 2.

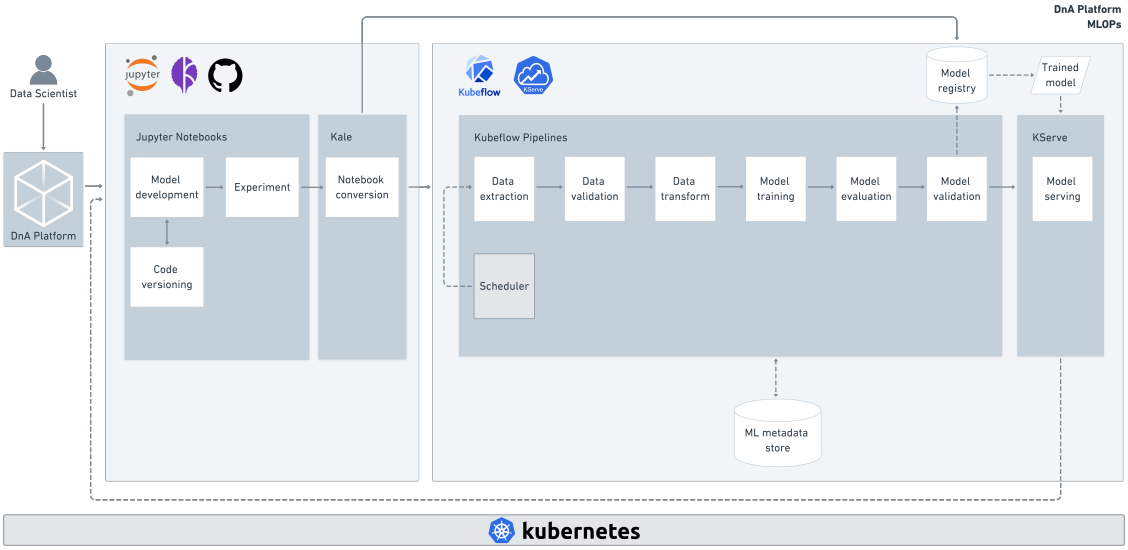

Figure 4.2: The MLOPs Workflow for the DnA Platform

5 Solution

The installation of the designated tools was a multi-step process. In the beginning, the main goal was to demonstrate their feasibility, explore and test their features, and identify possible challenges and risks. Thus, for what is known as Proof of Concept (PoC), JupyterHub, KFP, and KServe were deployed locally in a Docker-Desktop Kubernetes cluster. This procedure required relatively short time and effort, as these tools, despite their recent introduction to the technological world, had managed to attract several interested parties who have already published some installation guides online. However, quite often, these guides were missing crucial information, or they were not complete. That led to some deeper exploration and debugging. Nonetheless, the local deployment helped significantly in the PoC, and in the end, it acted as a demo of how the MLOps workflow can work inside the DnA platform. Since the online coverage of the individual steps was not enough at that time (November 2021), in Appendix A, one can find all the necessary files and commands to set up the tools locally.

Then, the following steps were oriented around the setup of the MLOps tools in the enterprise Kubernetes cluster of Mercedes-Benz AG. Initially, a plan was created to prioritize the multiple tasks. The first major assignment was the set up of KFP, then the installation of Kale in the JupyterHub of the DnA platform and the deployment of KServe. In the end, the most important job was to flawlessly connect and integrate them into the platform to achieve the objective of creating an open-source MLOps experience.

This chapter presents the challenges that occurred during the installation process in the enterprise environment and the solutions to the respective problems.

5.1 Deploying KFP on an Enterprise Cluster

The local installation of Kubeflow Pipelines is a relatively smooth process that one can complete in a matter of time by applying two sets of customized manifests (as described in Appendix A). However, the attempt to set up KFP on the Kubernetes cluster of any enterprise, in this case in the security-oriented Mercedes-Benz AG, is rather challenging. This section describes the main problems that ensued during this process and the modifications implemented to overcome them.

5.1.1 KFP Version and Kubernetes Manifests

Initially, the standalone deployment manifests set of Kubeflow Pipelines was selected because the goal was to install the minimum components from the Kubeflow ecosystem. That proved to be particularly handful in analyzing and tackling all the challenges related to the KFP implementation in an enterprise environment. Nonetheless, the standalone KFP deployment was not adequate as even the most current version of 1.7.0 (December 2021) doesn't provide the multi-user isolation feature. Hence, more Kubeflow components were required to achieve the multi-tenancy. The installation of Kubeflow Pipelines in the Mercedes-Benz environment can be divided into two distinct parts:

1. the deployment of common components used in different Kubeflow projects,
2. the deployment of the required for the multi-user version of KFP components, and
3. the deployment of the multi-user KFP.

This implementation includes the most current and stable versions (Figure 5.1) of the different components. Following is presented a comprehensive overview of each module's functionality and practicality:

1. Common components
    - *Cert-Manager* was already installed in the DHC CaaS cluster as it is utilized not only from Kubeflow but also from other applications. In general, this tool is a Kubernetes add-on, responsible for issuing, managing, validating, and updating TLS certificates from several issuers such as ACME or SelfSigned.

Kubeflow Components

Component	Version	Required
Training Operator	1.4.0	
MPI Operator	1.4.0	
Notebook Controller	1.4.0	
Tensorboard Controller	1.4.0	
Central Dashboard	1.4.0	✓
Profile + KFAM	1.4.0	✓
PodDefaults Webhook	1.4.0	✓
Jupyter Web App	1.4.0	
Volumes Web App	1.4.0	
Katib	0.12.0	
KServe	0.6.1	
Kubeflow Pipelines	1.7.0	✓
Kubeflow Tekton Pipelines	1.0.0	

Common Components

Component	Version	Required
Cert-Manager	1.3.1	✓
Istio	1.9.6	✓
Knative	0.22.1	

Figure 5.1: The components used for the KFP installation

- *Istio and Kubeflow Istio Resources*. Istio was already deployed and used in the DHC CaaS cluster by numerous applications. It is one of the most popular and widely adopted open-source service meshes. More particularly, it acts as an additional service networking layer that enables network authorization, routing policies, and secure connectivity between different microservices. Additionally, the Kubeflow Istio Resources is a configured set of manifests that creates the Istio resources required by Kubeflow. Mainly it generates an Istio Gateway called Kubeflow-Gateway, which is essentially a load balancer to distribute network traffic between the Kubeflow components.
- *OIDC Auth Service* enables the usage of an OIDC client by extending the Istio Ingress-Gateway. That was crucial for the multi-user version of KFP because Mercedes-Benz has its own custom-made OIDC Provider to allow employees accessing the internal services. Besides, since multi-tenancy is required, the authentication of users is a must. The OIDC Auth Service grants the utilization of any OIDC provider that implements the OAuth 2.0 protocol.
- *Kubeflow Roles* is a set of manifests that creates the Kubeflow ClustersRoles used for the different user permission levels. There are three different ClusteRoles: view, edit, and admin.

2. Multi-User Kubeflow Pipelines dependencies

   - *Central Dashboard* is the UI Kubeflow component. While typically, each Kubeflow tool has its UI element (i.e., Kubeflow Pipelines UI), the Central Dashboard acts as a housing for the different UIs (Figure 5.2). In addition, the home page provides an overview of the user's operations and several shortcuts to internal (for instance, recent runs) and external (for example, documentation pages) sources. The advantage of this component is that it is modular and easily configurable, which means that the modules that are not used (for instance, Katib) can be removed from the UI. Last but not least, the home page can be adjusted to include shortcuts to desired destinations.
   - *Admission Webhook* is used to modify or inject default specs (env vars, volumes) to pods. In particular, Kubeflow provides several PodDefault manifests describing runtime requirements (for example, certificates) that must be infused into a pod when spawned. The PodDefault manifests specify the Pods to which they apply. When a pod creation is requested, the Admission Webhook searches for the proper PodDefault manifest (if any) and mutates the Pod specs accordingly.
   - *Kubeflow Profiles Controller* is the fundamental component used by Kubeflow to achieve the desired multi-user isolation. More specifically, multi-tenancy is accomplished firstly by namespaces in which a user or a group of users are isolated and secondly by Profiles which are unique configurations per user. The Kubeflow Profile Controller is responsible for every profile:
     - for managing the RBAC RoleBinding to define the namespace administrator
     - for managing the Istio namespace-scoped ServiceRole and ServiceRoleBinding, which allows the profile owner to access services inside the namespace via Istio
     - for setting up the editor and viewer Service Accounts used by the user-spawned pods

- *Kubeflow Access-Management (KFAM)* allows self-serving Kubeflow by giving permissions to each user to create, automatically via UI, its profile, therefore namespace. While this option was pre-enabled initially, it is usually not desired in an enterprise environment for the reasons described in 5.1.6. Eventually, it was disabled in the Mercedes-Benz AG environment.

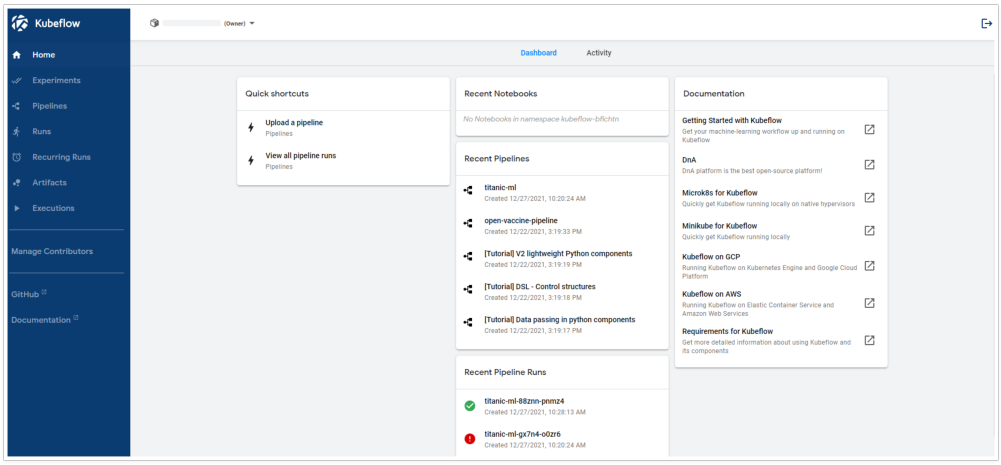

Figure 5.2: Example of the Kubeflow Dashboard

3. Multi-user Kubeflow Pipelines components

   - *API Service* is used to build, run and manage pipelines via a REST API. The operations definitions and descriptions are thoroughly reported on the official Kubeflow documentation page (Kubeflow-authors, 2021).

   - *Argo Workflow Controller* is the fundamental workflow execution engine for the Kubeflow Pipelines. It is an open-source tool that is particularly handy in orchestrating Kubernetes jobs to run in parallel like multi-step workflows, which essentially the Kubeflow Pipelines are. Kubeflow is offering for the integration with KFP a pre-configured Argo Workflow Controller config map. The main functionality of the tool derives from the workflow executor, a process that follows a defined interface to allow Argo to execute tasks such as monitoring pod logs, collecting artifacts, managing container lifecycles, and so on. Argo Workflow supports various workflow executors types. While the most popular one, and the default until version 3.2, was the Docker executor, in June 2021, a new executor, Emissary, was released. Its main advantage is that it is more

secure than the others, as it does not require privileged or root access. The Emissary executor became the new default option in the Argo Workflow engine because security is considered more important, despite its current lower reliability compared to the Docker executor. The KFP version installed in the DHC CaaS cluster uses the Docker-executor as default. Nonetheless, it also provides an easy way to replace it with the Emissary one. As a result, and due to security restrictions in the Mercedes Benz environment, where no tool is allowed to run with root privileges, the emissary executor got chosen for the Argo Workflow Controller.

- *Cache Deployer and Server* provide one of the most crucial features of KFP, step-level caching. Each time a pipeline is running, KFP validates the existence or the executed step. If the task is already complete from a previous run, then the computation of the particular part is skipped to reduce computational and time costs, and the results are loaded from the cache. Alternatively, if the step is for the first time executed, its execution gets cached. This feature is by default enabled for all tasks.
- *Pipelines Profile Controller* is in charge of deploying in the user namespace, every time a profile is generated, two plugins, the UI Artifact and the Visualization Server. The two plugins are used to visualize the output produced from the execution of the pipeline components.
- *Pipelines UI* enables users to perform several operations related to KFP. More specifically, through the Pipelines UI, which is accessible via the Central Dashboard (Figure 5.2), users can enter the following tabs:
  - Experiments tab that contains groups of one or more pipeline runs and the ability to create new experiments
  - Pipelines tab that contains the definitions and the different versions of generated pipelines and the ability to upload a new pipeline
  - Runs tab that contains the pipelines that already run and the ability to create a new run

  - Recurring Runs tab that contains the recurring pipeline runs and the ability to schedule a new run
  - Artifacts tab that contains the outputs emitted by an executed pipeline
  - Executions tab that contains information about executed pipeline steps

- *Metacontroller* is a widely used Kubernetes add-on that enables the easy creation and deployment of custom Kubernetes controllers (Metacontroller, 2021). For the KFP, the Metacontroller is used to define the Pipelines Profile Controller.
- *Metadata* components are used to record and retrieve metadata from the execution of pipelines. Data such as the Artifacts, Executions, Events, and more are written in and loaded via the Metadata Store from the storage backend. This feature is crucial for the KFP, and it provides numerous benefits. For instance, determining if execution has run with the same data input in the past, identifying all the artifacts generated from a specific artifact, comparing two artifacts, etc.
- *MinIO* serves as the default storage option for the KFP artifacts. Namely, the pipeline packages, views, and metrics. It is worth mentioning that Kubeflow supports different storage types, including s3, on-prem, etc.
- *MySQL* is utilized as the database for the KFP metadata, such as experiments, jobs, pipeline runs, etc.
- *Pipeline Persistence Agent* monitors and preserves the resources generated by the Pipeline Service in the Metadata Service. Additionally, this module records the parameters and the data artifact URIs of the executed containers.
- *Pipeline Scheduled Workflow* is used for creating recurring pipeline runs. Those runs can be adjusted by a run trigger, for instance, a periodic trigger to run the pipeline every 1 hour. In addition, the recurring runs allow the user to specify the run required parameters.
- *Pipeline Viewer Controller* is responsible for the web views management of instances like the Tensorboard ones directly in the Pipelines UI.

5.1.2 Public Registry Docker Images

The Kubeflow components are packed into Docker images, which come with multiple and already widely known advantages such as isolation, mobility, flexibility, modularity, etc. Those Docker images are stored in a public container registry named Google Cloud Registry to enable open-source users around the globe to use them freely when installing Kubeflow. Nevertheless, these registries are usually not used in an enterprise environment due to potential security and privacy flaws. Most of the organizations, like Mercedes-Benz, have their own private registries to control which images are stored where and apply custom configuration options such as authentication, logging, etc. The Docker images used in the selected Kubeflow manifests were pulled, checked for potential security issues, and pushed to the private Docker registry of Mercedes-Benz, called Harbor, to overcome this problem. The Docker images list can be found in Figure 5.3.

Public Registry Docker Images

gcr.io/arrikto/kubeflow/oidc-authservice	gcr.io/ml-pipeline/metadata-envoy
docker.io/istio/pilot	gcr.io/ml-pipeline/metadata-writer
gcr.io/istio-release/proxyv2	docker.io/minio/minio
public.ecr.aws/j1r0q0g6/notebooks/access-management	gcr.io/ml-pipeline/mysql
public.ecr.aws/j1r0q0g6/notebooks/admission-webhook	gcr.io/ml-pipeline/persistenceagent
public.ecr.aws/j1r0q0g6/notebooks/central-dashboard	docker.io/python
public.ecr.aws/j1r0q0g6/notebooks/profile-controller	gcr.io/ml-pipeline/scheduledworkflow
gcr.io/kubebuilder/kube-rbac-proxy	gcr.io/ml-pipeline/viewer-crd-controller
docker.io/metacontroller/metacontroller	gcr.io/ml-pipeline/visualization-server
gcr.io/ml-pipeline/api-server	gcr.io/ml-pipeline/workflow-controller
gcr.io/ml-pipeline/application-crd-controller	gcr.io/tfx-oss-public/ml_metadata_store_server
gcr.io/ml-pipeline/argoexec	gcr.io/ml-pipeline/cache-deployer
gcr.io/ml-pipeline/frontend	gcr.io/ml-pipeline/cache-server
gcr.io/google-containers/busybox	

Figure 5.3: The public images used in this deployment

5.1.3 Non-root Installation

One of the most crucial drawbacks behind the Kubeflow deployment is the lack of security design. The majority of Kubeflow components are by default requesting advanced privileges, something which in a real-life enterprise environment is in any case unacceptable. During the project, the challenge of transforming the numerous Kubeflow modules into secure setups without affecting their functionality was the most demanding and

time-consuming process. In the beginning, the installation in the DHC CaaS cluster took place with the original manifests. As the security policy applied in Mercedes-Benz is intensely strict, the issue was spotted it immediately. The containers couldn't be spawned at all, as they were based on Docker images requesting root privileges (Figure 5.4). The solution to mitigate this problem had to be abstract and applicable to as many parts as possible to minimize the effort and the functionality affection.

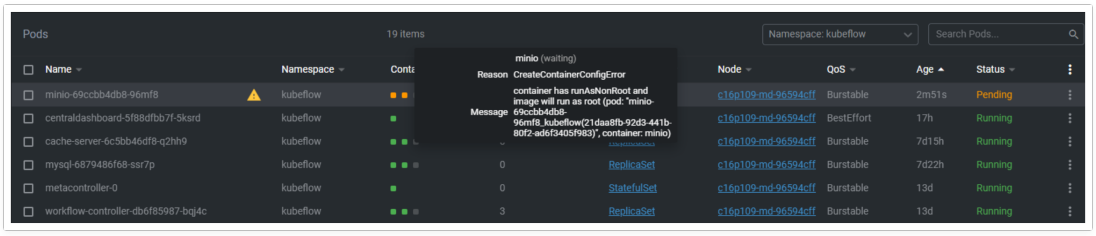

Figure 5.4: Example of the error message when using the default MinIO manifest

The most suitable way to configure a Pod's or Container's privilege and access control settings is via Kubernetes. Kubernetes provides the Security Context feature that enables the application of a constraint set to a container. By principle, the Security Context is used to lessen the security risks inside a system with multiple deployments. Similarly, for the KFP installation in the DHC CaaS cluster, the Security Context was added to run all the services as distinct non-root users. While there are provided at least ten different setting options by Kubernetes (Smalling, 2021), the following were sufficient for the project:

1. *runAsNonRoot* is a boolean Security Context setting to declare if the container of a pod should run as a non-root user. If true, then during runtime, kubelet will check if the Docker image requires to run as root (UID = 0) and if so, it won't let the container start (Goltsman, 2019).

2. *runAsUser*, determines the user, via the User ID (UID), to run the container Entrypoint and the processes within. If this setting is not applied then, the default user of the image will be used. For security reasons, the user with ID 0 (root user) shouldn't be used (Goltsman, 2019).

3. *runAsGroup*, specifies the primary group, via the Group ID (GID), to run the pro-

cesses inside a container. If this setting is not applied then, the default group of the image will be used. The group with ID 0 (root) shouldn't be used (Kubernetes-authors, 2021).

4. *fsGroup* declares the group, via the group ID (GID), that will be the owner of the filesystem and every new file added in every container inside a pod. Concurrently, the fsGroup Security Context specifies the owner of any volume mounted and thus who can write to it (Smalling, 2021).

The settings were applied to all the modules to reduce the security risks. Nevertheless, it must be noted that in many cases, different UIDs or GIDs were used, depending on the default individual Docker images users (Listing 5.1). For the Docker images where the default UID and GID were impossible to detect or not declared, the user and group *nobody* (ID = 65534) were selected.

Listing 5.1: The user 8737 was selected because this ID is the default in the argo-workflow Docker image

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: workflow-controller
spec:
  selector:
    matchLabels:
      app: workflow-controller
  template:
    metadata:
      labels:
        app: workflow-controller
        spec:
      securityContext:
        runAsNonRoot: true
        fsGroup: 8737
        runAsGroup: 8737
        runAsUser: 8737
...
```

To conclude, transforming the Kubeflow installation into a secure setup was demanding, required a lot of exploration and try-and-error processes. This fact already hints that despite the promising core idea behind the tool and the, in general, well-organized open-source community support, Kubeflow is not in any case ready for enterprises purposes. Nevertheless, all the implemented security changes are planned to be contributed to the Kubeflow community to judge if they are acceptable or if they can furtherly be optimized and finally adopted.

5.1.4 Certificate Signing Requests

Caching is one of the most promising features of KFP. Each time a pipeline step is the same as an already executed, the results are loaded from the cache server. Therefore, there is no need to re-run the whole pipeline each time a part of the ML model gets adjusted. Undoubtedly, caching can lead to crucial cost and time reductions, making the MLOps lifecycle more efficient. As described in 5.1.1, caching is accomplished in KFP via two interdependent modules: the *cache deployer* and the *cache server*. The naming already indicates that the *cache deployer* is the module responsible for deploying the *cache server*.

While trying to set up the modules in the DHC CaaS cluster, it was noted that the installation couldn't be completed. The reason was that the *cache deployer* is built to generate a Signed Certificate for the *cache server* by referring to the Kubernetes CertificateSigningRequest API (Listing 5.2).

Listing 5.2: Original K8s CSR

```
...
cat <<EOF >> ${tmpdir}/csr.conf
[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[req_distinguished_name]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
```

```
subjectAltName = @alt_names
[alt_names]
DNS.1 = ${service}
DNS.2 = ${service}.${namespace}
DNS.3 = ${service}.${namespace}.svc
EOF

openssl genrsa -out ${tmpdir}/server-key.pem 2048
openssl req -new -key ${tmpdir}/server-key.pem -subj "/CN=${service}.${
    namespace}.svc" -out ${tmpdir}/server.csr -config ${tmpdir}/csr.
    conf

echo "start running kubectl..."

# clean-up any previously created CSR for our service. Ignore errors if
     not present.
kubectl delete csr ${csrName} 2>/dev/null || true

# create server cert/key CSR and send to k8s API
cat <<EOF | kubectl create -f -
apiVersion: certificates.k8s.io/v1beta1
kind: CertificateSigningRequest
metadata:
  name: ${csrName}
spec:
  groups:
  - system:authenticated
  request: $(cat ${tmpdir}/server.csr | base64 | tr -d '\n')
  usages:
  - digital signature
  - key encipherment
  - server auth
EOF

# verify CSR has been created
```

```
while true; do
   kubectl get csr ${csrName}
   if [ "$?" -eq 0 ]; then
      break
   fi
   sleep 1
done

# approve and fetch the signed certificate
kubectl certificate approve ${csrName}
# verify certificate has been signed
for x in $(seq 10); do
   serverCert=$(kubectl get csr ${csrName} -o jsonpath='{.status.
      certificate}')
   if [[ ${serverCert} != '' ]]; then
      break
   fi
   sleep 1
done
if [[ ${serverCert} == '' ]]; then
   echo "ERROR: After approving csr ${csrName}, the signed certificate
      did not appear on the resource. Giving up after 10 attempts."
      >&2
   exit 1
fi
echo ${serverCert} | openssl base64 -d -A -out ${tmpdir}/server-cert.
   pem

echo ${serverCert} > ${cert_output_path}

# create the secret with CA cert and server cert/key
kubectl create secret generic ${secret} \
      --from-file=key.pem=${tmpdir}/server-key.pem \
      --from-file=cert.pem=${tmpdir}/server-cert.pem \
      --dry-run -o yaml |
```

```
kubectl -n ${namespace} apply -f -
```

The usage of API server certificates in the Mercedes-Benz AG is restricted because those allow permission escalation. The security risk is critical, as by using this API, users can order certificates that let them impersonate both Kubernetes control plane and DHC CaaS team access. The goal was to adjust the *cache deployer's* certificate generation process without affecting the actual functionality to avoid loosening the security restrictions. With the help of the DHC CaaS administrator team, it was pointed out that the Kubernetes CSR API usage could be exchanged with CSRs generated by the widely-known OpenSSL. Indeed, the changes were implemented inside the script that is responsible for the CSR creation (Listing 5.3), and the *cache deployer* Docker image was adjusted accordingly to use this script. With this optimization, installing the caching tools was accomplished in a more secure and plain, in terms of coding lines, manner. The implemented security changes are planned to be contributed to the Kubeflow community to judge if they are acceptable or if they can furtherly be optimized and finally adopted.

Listing 5.3: OpenSSL CSR

```
...
cat <<EOF >> ${tmpdir}/csr.conf
[req]
x509_extensions = v3_req
distinguished_name = req_distinguished_name
[req_distinguished_name]
[v3_req]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names
[alt_names]
DNS.1 = ${service}
DNS.2 = ${service}.${namespace}
DNS.3 = ${service}.${namespace}.svc
EOF
```

```
# Due to security constraints we cannot use the Kuberentes CSR,
    therefore we create an openssl certificate
openssl req -x509 -nodes -newkey rsa:4096 -keyout $tmpdir/server-key.
    pem -out $tmpdir/server-cert.pem -sha256 -days 3650 -config ${
    tmpdir}/csr.conf -subj "/CN=${service}.${namespace}.svc"

openssl base64 -in ${tmpdir}/server-cert.pem -A -out ${cert_output_path
    }

# create the secret with CA cert and server cert/key
kubectl create secret generic ${secret} \
      --from-file=key.pem=${tmpdir}/server-key.pem \
      --from-file=cert.pem=${tmpdir}/server-cert.pem \
      --dry-run -o yaml |
   kubectl -n ${namespace} apply -f -
```

5.1.5 Exposing the KFP UI

The default way to access the Kubeflow UI is via port-forwarding of the Istio - Ingress Gateway service, created after the installation. In addition, the documentation explicitly specifies that to connect to Kubeflow from outside the Kubernetes cluster, HTTPS must be set up because Secure Cookies are used from various components. In the case of the DHC CaaS cluster, a sub-host with SSL encryption to enforce HTTPS was created, and the Istio - Ingress Gateway service was exposed via an Ingress load balancer (Listing 5.4).

Listing 5.4: Kubeflow Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: kubeflow-ingress
  annotations:
    traefik.frontend.rule.type: PathPrefix
    kubernetes.io/ingress.class: traefik
    traefik.ingress.kubernetes.io/router.tls: "true"
```

```
    traefik.ingress.kubernetes.io/router.entrypoints: websecure
    cert-manager.io/cluster-issuer: ******-****
spec:
  rules:
    - host: kfp.dna-dev.app.*******.net
      http:
        paths:
          - backend:
              service:
                name: istio-ingressgateway
                port:
                  number: 80
            path: /
            pathType: Prefix
  tls:
    - hosts:
      - kfp.dna-dev.app.*******.net
      secretName: kfp-tls-secret
```

5.1.6 OIDC & Multi-User Isolation

The main idea behind the DnA platform is to provide a set of tools to users to create, manage and share their data analytics-oriented solutions. Following this pattern, the goal of this thesis was to design and implement a complete MLOps pipeline within the DnA platform, which would enhance the user experience and reduce the technical debt. Since, DnA is scoped as a multi-user toolkit platform, enabling multi-user isolation is crucial for transparency and efficient usage of infrastructure and operations.

The first step towards multi-tenancy is the ability for users to connect to their own space, achieved by using authentication mechanisms. For consistency, many enterprises, like Mercedes-Benz, provide users a common way to sign in to the different organization services. Furthermore, authenticating with a single account is also used widely in public platforms. For instance, users can use their email account to sign in to other services, like a music or a cloud storage service. The most common practice around end-user

authentication promotes the utilization of the OAuth2 protocol by an OIDC provider. Mercedes-Benz AG uses the Mercedes-Benz AG Global Authentication System (GAS-OIDC), a custom OIDC provider, to enable a single sign-on between different company applications. Concurrently, the first step for the multi-tenancy in the KFP installation was to integrate it with the GAS-OIDC (Figure 5.5). Since Kubeflow utilizes the OIDC Auth Service described in 5.1.1, the connection between the GAS-OIDC and the KFP was accomplished by specifying the GAS-OIDC Endpoint, the redirect URL, the OIDC scopes, and the USERID claim as seen in the Listing 5.5. The OIDC Auth Service supports the integration with almost any OAuth2 OIDC provider. As a result, this process was straightforwardly implemented.

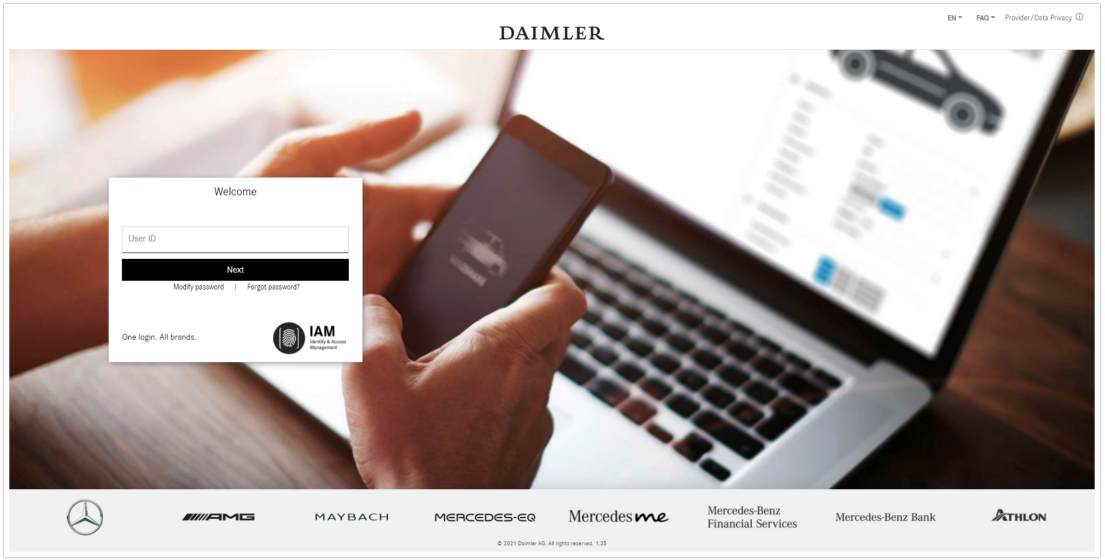

Figure 5.5: Mercedes-Benz AG Global Authentication System (GAS-OIDC)

Listing 5.5: KFP and GAS-OIDC integration

```
OIDC_PROVIDER=https://***-**.*****.com
OIDC_SCOPES=email
REDIRECT_URL=https://kfp.dna-dev.app.*******.net/login/oidc
USERID_HEADER=kubeflow-userid
USERID_PREFIX=
USERID_CLAIM=sub
PORT="8080"
STORE_PATH=/var/lib/authservice/data.db
```

The second step includes the implementation of the multi-user spaces. More specifically, Kubeflow supports multi-tenancy for several components such as the Central Dashboard, KFP, etc. The main module used for this operation, as described in 5.1.1, is the Kubeflow Profile controller. During the project implementation period (October 2021 - March 2022), Kubeflow provided two ways for creating a user's profile. On the one hand, there is the manual way, where an admin is responsible for creating each profile by applying in the Kubernetes cluster the required resources (RBAC, ServiceRole, etc.). As the DnA platform is oriented towards thousands of users, this option is not efficient nor productive, but at the same time, it ensures the strict and secure management of users. On the other hand, Kubeflow offers, via KFAM, an automatic way where users need to specify only the name of their profile, and the rest is taking place automatically in the background. It is worth mentioning again that the profile name corresponds to the namespace name created for the user. This option is more efficient for an environment with numerous users, but it lacks applying a specific policy around user-profiles and can lead to potential management and security issues. A characteristic example is that a user may want, for malicious reasons, to create a namespace called after another user's name. Furthermore, a disadvantage of this profile creation process is that profile management becomes particularly difficult because no naming policy is applied. As a result, the challenge was to design an automated process for the DnA platform, so users could get KFP profiles that would follow a specific policy and wouldn't involve the user or an admin in the process.

The main point behind the solution of this challenge for the DnA platform is to automatically create KFP profiles for users who spawn their JupyterNotebook instance. More specifically, as the DnA platform utilizes the JupyterHub as a service, customization is achievable via the hub configuration file. Via this file, the JupyterHub can be modified to spawn, in various ways customized, JupyterNotebooks. The hub configuration file was already used in the case of the DnA platform for a variety of settings. The overall architecture can be seen in Figure 5.6. Users log in to the DnA platform, and when they launch JupyterHub for the first time, a JupyterNotebook instance and a KFP profile are generated inside their custom namespace. For the KFP automatic profile generation, the following steps implemented:

1. Created a kustomization.yaml (Listing 5.6) to include all the necessary resources that, by design, the KFAM module would generate and apply during the automated profile creation by the user, such as the Kubeflow profile instance, RBAC, Service accounts, etc.

Listing 5.6: The kustomization.yaml used for the Kubeflow Profile generation

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- networking-policy.yaml
- profile-instance.yaml
- pod-default.yaml
- envoy-filter.yaml
- kale-workflow-rbac.yaml
- ml-pipeline-authorization.yaml
- notebooks-authorization.yaml
configMapGenerator:
- name: default-install-config
  namespace: kubeflow
  envs:
  - params.env
vars:
# These vars are used for substituing in the parameters from the config
    map
# into the Profiles custom resource.
- name: user
  objref:
    kind: ConfigMap
    name: default-install-config
    apiVersion: v1
  fieldref:
    fieldpath: data.user
- name: profile-name
  objref:
```

```
    kind: ConfigMap
    name: default-install-config
    apiVersion: v1
  fieldref:
    fieldpath: data.profile-name
- name: sa
  objref:
    kind: ConfigMap
    name: default-install-config
    apiVersion: v1
  fieldref:
    fieldpath: data.sa
configurations:
- params.yaml
```

2. An additional resource (Listing 5.7) of kind Authorization Policy, named notebooks-authorization, was created and included in the kustomization.yaml. The purpose of this manifest is to allow the admin of the custom namespace, hence the user, to access all the operations under the /notebooks/* path to spawn the Jupyter Notebook instance.

Listing 5.7: The additional resource for the kustomization.yaml

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: notebooks-authorization
  namespace: $(profile-name)
spec:
  action: ALLOW
  rules:
    - when:
        - key: source.namespace
          values:
            - $(profile-name)
```

```
- to:
  - operation:
      paths:
      - /notebooks/*
```

3. Since most of the above manifests are namespace-scoped, the goal was to make them on the fly adjustable. That is achieved by creating an empty params.env file that is being overwritten every time, based on the specific userid. The kustomization resources user variables that are referring to the params.env file. In this way, they can be adjusted easily in an automated process.
4. Next, the manifests were copied inside the JupyterHub Docker image.
5. The JupyterHub deployment was adjusted as follows:
   - More privileged RBAC designated to the hub Service Account to allow the application of the kustomization.yaml in the DHC CaaS cluster. Nevertheless, using the *c.KubeSpawner.service_account = "default-editor"* inside the hub config file (Listing 5.8), the default Service Account of the spawning JupyterNotebook modified to match the restricted privileges that the user has in the KFP profile. As a result, the spawned notebook remains secure and doesn't provide additional permissions to its user.
   - The notebook namespace should be the same as the Kubeflow Profile name. That is achieved by using inside the hub config file a KubeSpawner function called *user_namespace_template* (Listing 5.8).
   - Using the Kubespawner *pre_spawn_hook* in hub config, one can bootstrap work that would run just before the spawning of the notebook starts. This feature is crucial, as the goal is to apply the set of manifests that will create the Kubeflow Profile, hence the namespace before the generation of the notebook. A *pre_spawn_hook* function (Listing 5.8) was written to:
     - Overwrite the params.env file with the necessary user-based values
     - Apply the kustomization.yaml and its resources

Listing 5.8: The modified part of the hub-config.yaml

```
...

c.KubeSpawner.service_account = "default-editor"
c.KubeSpawner.user_namespace_template = "kubeflow-{username}"

#Create a per-user namespace and Kubeflow Profile
def profile_prespawn_hook(spawner):
   username = spawner.user.name
   namespace = f"kubeflow-{username}"
   sa = f"cluster.local/ns/{namespace}/sa/default-editor"
   # overwrite the params.env file with username and namespace
   with open("/tmp/kfp-user-namespace/params.env", "w") as file:
      file.write(f"user={username.upper()}\nprofile-name={namespace}\
          nsa={sa}\n")

   cmd_kfp = f"kubectl apply -k /tmp/kfp-user-namespace"

   subprocess.run(cmd_kfp, shell=True, check=True)

c.KubeSpawner.pre_spawn_hook = profile_prespawn_hook

...
```

In conclusion, the desired multi-tenancy is achieved by integrating the KFP deployment with the GAS-OIDC and creating an automated, secure, and enterprise-ready KFP Profile creation process. Users can log in and use their MLOps instances, which are being safely and without any effort generated in their namespace. Additionally, the DnA platform administrators' endeavor is eliminated as their interaction in the process is not required at all. The current implementation contributes to one of the main scopes of this thesis: the reduction of the technical dept and the user experience improvement.

It must be noted, though, that the latest version of KFP used in this project (December 2021) doesn't support isolation for:

1. Pipeline definitions
2. Artifacts, executions, and other metadata entities

The overcoming of these limitations is already a work in progress for the future releases of Kubeflow.

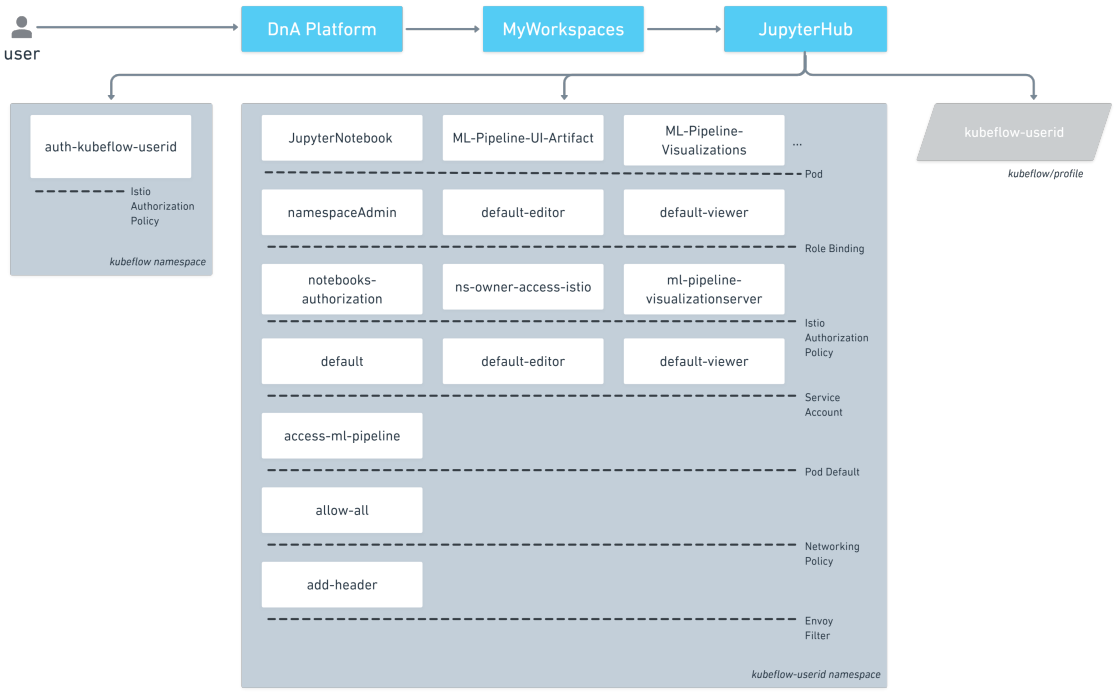

Figure 5.6: Architecture of the KFP profile generation process

5.1.7 Integration with the DnA Platform

Initially, the KFP UI was exposed as described in 5.1.5. Nevertheless, the integration with the DnA platform was crucial, as the goal was to create a complete MLOps pipeline that would enhance the user experience. The DnA users can access the KFP UI in two different ways.

On the one hand, the access can happen directly from the Jupyter Notebooks via the Kale UI. One of the principal features of this project is the flawless integration of the tools used in the MLOps pipeline. Users develop their models in the Jupyter Notebooks, and then, using Kale, they can create, upload and run a Kubeflow Pipeline effortlessly from the same environment. As seen in Figure 5.7, the Kale UI generates two hyperlinks during the uploading and running of a Kubeflow Pipeline. By clicking on them, users are

directly transferred to the KFP UI and access the relevant information about the specific pipeline.

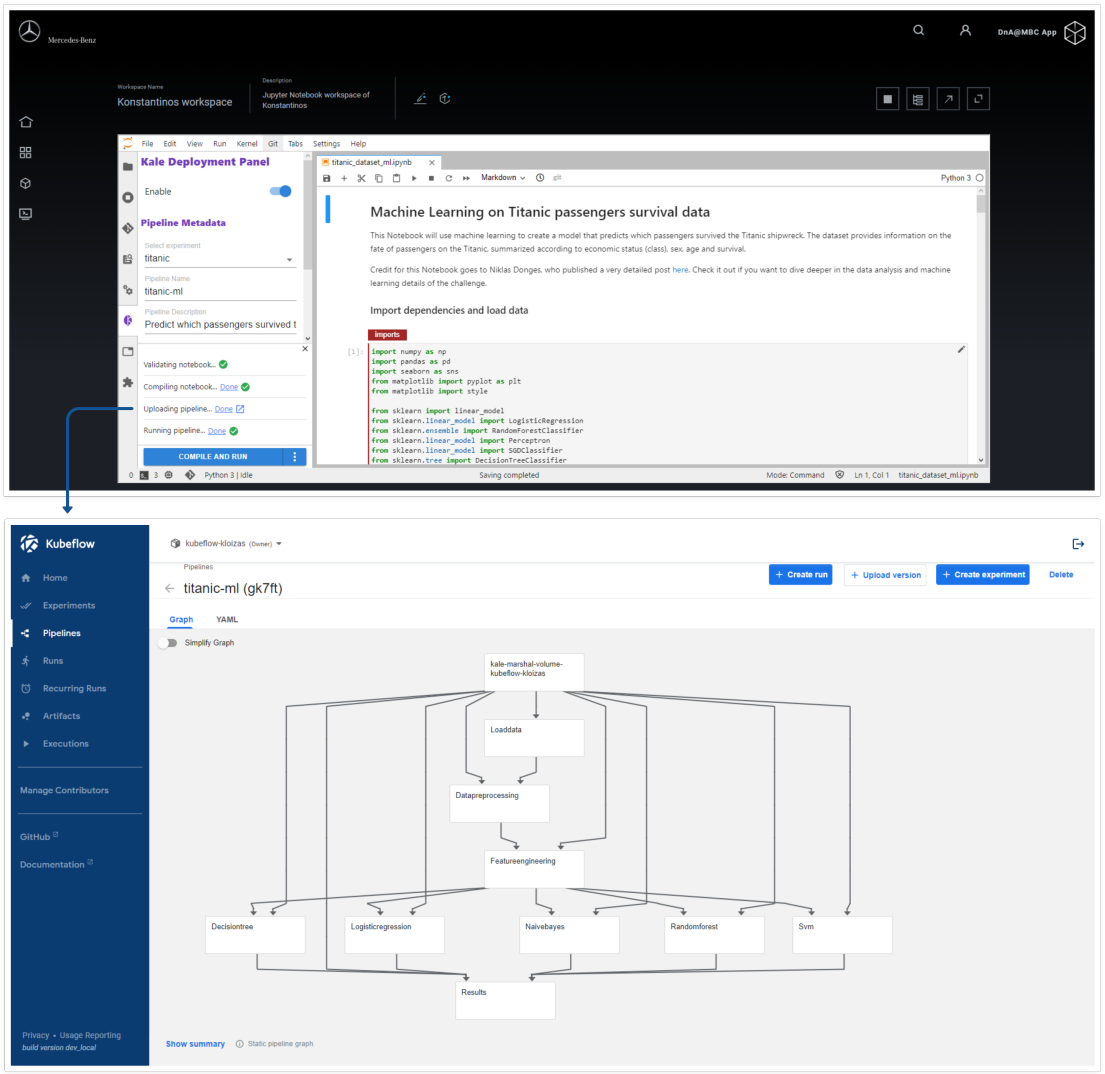

Figure 5.7: Kale generates hyperlinks to access the KFP UI

On the other hand, the KFP UI should be accessible without uploading or running a Kubeflow Pipeline each time. Consequently, a new tile was developed in the My Services section of the platform (Figure 5.8). From there, users can access the KFP UI directly. The DnA team plans to integrate the KFP UI, in a future release, inside the DnA platform, such as Jupyter Notebooks are embedded.

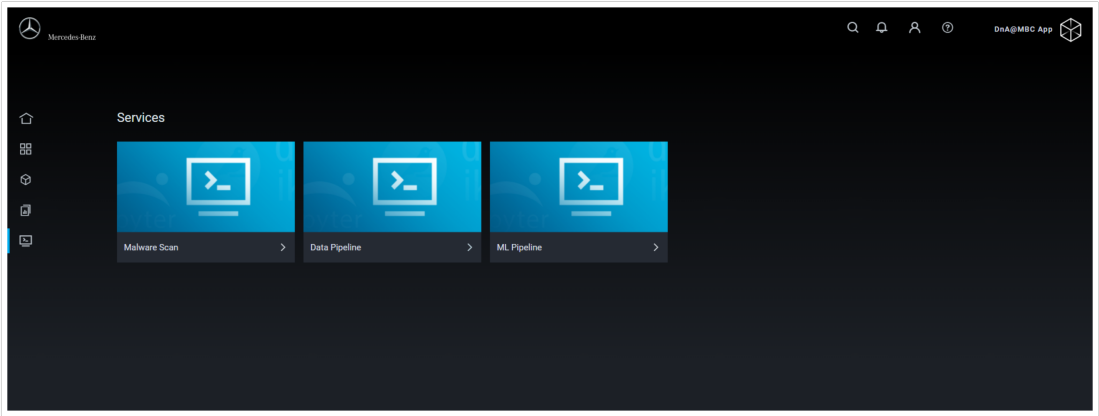

Figure 5.8: Accessing the KFP UI via the My Services section

5.2 Deploying Kale on an Enterprise Cluster

The local installation of Kale is an adequately easy process, as described in Appendix A. Nonetheless, the setup of the tool in the existing JupyterHub installation of the DnA platform and its connection to the KFP was a process that required several modifications. This section describes all the implemented adjustments.

5.2.1 Kale Version and JupyterLab Extension

For the installation of Kale, the most recent and stable version got selected (v.0.7.0). In particular, the Kale backend and the JupyterLab extension are available from the Python Package Index (PyPI) repository. However, in order to have maximum flexibility around the customization of the tool, Kale is installed directly from the GitHub repository. Specifically, the Kale branch is cloned inside the JupyterLab single-user notebook Dockerfile. Then, the setup takes place in three steps:

1. Installation of the required dependencies:
    - For the selected version of Kale, the installation of the KFP SDK, which is included in the source code, requires a specific version of the python library enum34.
    - One of the considerable drawbacks of Kale is that even its most updated version (December 2021) supports version 2.x. of JupyterLab, while the most cur-

rent one is version 3. The security impact was assessed, and after communication with the open-source community behind the development of Kale, it was communicated that the support for the newer versions of JupyterLab is on the team's roadmap (Katsakioris, 2021).

2. Installation of the Kale backend:
    - The necessary backend files are loaded from the tool's source code.
3. Installation of the Kale lab extension:
    - The necessary extension files are loaded from the tool's source code.

In general, the above steps make the installation of Kale straightforward. Nevertheless, the tool's functionality in the secure environment of the Mercedes-Benz cluster is possible only after the modifications described in the following sections.

5.2.2 Non-root Installation

Kale is a very handful tool designed to automate the generation of Docker containers for each KFP component. Nonetheless, all the Kale versions are developed to spawn the different containers by setting a Security Context. More specifically, every time a user compiles a notebook using the tool, Kale generates a script according to which each container can be run only from a user with root privileges. As expected, this can lead to possible security incidents. Especially in an enterprise environment, running applications as root is considered a bad practice and highly prohibited. In the Mercedes-Benz environment, the DnA platform cluster management responsible team, DHC CaaS, is applying a harsh policy regarding user permissions and security. When firstly attempted to run a KFP generated by Kale, none of the pods could run. Hence, the pipeline was unexecuted. The log analysis pointed out explicitly that the pods couldn't run because: *the container has runAsNonRoot, and the image will run as root*. Modifications to the tool's backend were required to eliminate the security impact. In particular, Kale is using three jinja2 templates to generate the script for each KFP:

1. *nb_function_template*
2. *pipeline_template*
3. *py_function_template*

Listing 5.9: Original (Kubeflow-Kale, 2021) Kale Security Context

```
...

_kale_{{ step.name }}_task.container.set_security_context(k8s_client.
    V1SecurityContext(run_as_user=0))

...
```

In the *pipeline_template* (Listing 5.9), one can observe that for each KFP step, Kale sets as running user the UID 0, which is the ID of the root user. Initially, the solution is to modify the Security Context to any user with a UID > 0. In the case of the DnA platform, the Security Context adjusted as follows:

Listing 5.10: Secure Kale

```
...

_kale_{{ step.name }}_task.container.set_security_context(k8s_client.
    V1SecurityContext(run_as_user=8737, run_as_group=8737))

...
```

The user and group 8737 are chosen to match the pod Security Context, which is already set for the workflow controller of the Kubeflow Pipelines (5.1). Finally, the necessary change of the template is taking place during the installation of Kale in the Jupyter Notebooks. In particular, the new security-oriented template overwrites the original inside the Docker image where installation takes place. As a result, Kale generates a script where indeed, the KFP Security Context is secure, and the Kubeflow Pipeline containers can run in an enterprise environment without potential harm.

5.2.3 Kale and KFP Connection

Kale uses the Kubeflow Pipelines SDK to generate, upload and run pipelines. The connection between the two modules requires the KFP SDK client (Listing 5.11).

Listing 5.11: Example of the KFP SDK Client usage

```
import kfp
# When not specified, host defaults to env var KF_PIPELINES_ENDPOINT.
client = kfp.Client()
print(client.list_experiments())
```

In particular, by defining two env variables, Kale can communicate with the KFP deployment. These variables specify the endpoints of the *ml-pipeline* and the *ml-pipeline-ui* services. The setting of the endpoints was implemented inside the hub-config file mentioned in 5.1.6. There, JupyterHub enables the definition of environment variables via the Kubespawner function described in Listing 5.12.

Listing 5.12: Defining the KFP SDK client env variables inside the hub-config file

```
...
c.KubeSpawner.environment.update(
  {
   "KF_PIPELINES_ENDPOINT": "http://ml-pipeline.kubeflow:8888",
   "KF_PIPELINES_UI_ENDPOINT": "http://ml-pipeline-ui.kubeflow:80"
  }
)
...
```

In conclusion, the endpoint env variables are read by the KFP SDK client, and therefore the connection between Jupyter Notebooks, Kale, and KFP is established without any explicit arguments.

5.2.4 Marshal Volume and Artifacts Saving

The first step in every Kubeflow pipeline generated by Kale is the marshal volume's creation. The marshal volume, mounted in each pipeline step, is responsible for the data

serialization. Those are the data that pass between the several pipeline steps.Executing the different pipeline modules as the root user enables writing and reading data inside the /marshal directory. However, escalated user privileges in an enterprise environment are restricted, as described in 5.2.2. The pipeline steps are run by a non-root user. As a result, the serialization of the data cannot be achieved, and the generated Kubeflow Pipeline will fail. The solution was to pre-create the /marshal directory in the Docker image used for executing the pipeline and change the access permissions. This was achieved by using the following Unix chmod command:

chmod a=rwx,u+t /marshal

The command can be interpreted: Read, write, and execute permissions for the /marshal directory are given to everyone. Nevertheless, the sticky bit (*u+t*) ensures that files inside the /marshal directory may be renamed or removed only by their owner. Consequently, the non-root user used in each pipeline step has the required permissions to use the /marshal directory.

Listing 5.13: The original marshal volume naming

```
...
   {% if marshal_volume %}
   _kale_marshal_vop = _kfp_dsl.VolumeOp(
      name="kale-marshal-volume",
      resource_name="kale-marshal-pvc",
      modes={{ pipeline.config.volume_access_mode }},
      {%- if pipeline.config.storage_class_name %}
      storage_class="{{ pipeline.config.storage_class_name }}",
      {%- endif %}
      size="1Gi"
   )
...
```

Additionally, Kale uses, as described in 5.2.2, specific templates to generate the pipeline scripts. There, the name of the marshal volume is defined to be the same for each user (Listing 5.13). That can create several conflicts in a multi-user environment, especially when an operation looks for the user's specific persistent volume. A modification was

implemented in the naming policy of the marshal volume to avoid this issue. More specifically, an extra attribute was attached to include the namespace name (Listing 5.14). As the namespace is unique for each user (Section 5.1.6), this solves the problem of referencing the user-owned marshal volumes during the pipeline execution.

Listing 5.14: The modified marshal volume naming

```
...
   {% if marshal_volume %}
   _kale_marshal_vop = _kfp_dsl.VolumeOp(
      name="kale-marshal-volume-{{ current_namespace }}",
      resource_name="kale-marshal-pvc-{{ current_namespace }}",
      modes=['ReadWriteOnce'],
      {%- if pipeline.config.storage_class_name %}
      storage_class="{{ pipeline.config.storage_class_name }}",
      {%- endif %}
      size="500Mi"
   )
...
```

Last but not least, an additional modification was implemented in the script templates. More specifically, the pipeline metadata files are defined, by Kale, to be written in a /tmp folder to ensure that non-root users can write or update them (Listing 5.15). Nevertheless, this was not the case with the generated artifact files. Artifacts were saved on a different folder than required root privileges. Following the metadata example, the default directory was changed accordingly (Listing 5.16).

Listing 5.15: The original Artifacts saving directory

```
...
 {%- if autosnapshot %}
   _kale_output_artifacts.update({'mlpipeline-ui-metadata': '/tmp/
      mlpipeline-ui-metadata.json'})
   {%- endif %}
   {%- if step.metrics %}
   _kale_output_artifacts.update({'mlpipeline-metrics': '/tmp/
```

```
      mlpipeline-metrics.json'})
   {%- endif %}
   {%- if pipeline.processor.id == "nb" and step.name != "
      final_auto_snapshot" and step.name != "pipeline_metrics" %}
   _kale_output_artifacts.update({'mlpipeline-ui-metadata': '/tmp/
      mlpipeline-ui-metadata.json'})
   _kale_output_artifacts.update({'{{ step.name }}': '/{{ step.name }}.
      html'})
   {%- endif %}
   {%- if pipeline.processor.id == "py" and step.artifacts and step.
      name != "final_auto_snapshot" and step.name != "pipeline_metrics
      " %}
   _kale_output_artifacts.update({'mlpipeline-ui-metadata': '/tmp/
      mlpipeline-ui-metadata.json'})
   {%- for artifact in step.artifacts %}
   _kale_output_artifacts.update({'{{ artifact["name"] }}': '{{
      artifact["path"] }}'})
   {%- endfor %}
   {%- endif %}
...
```

Listing 5.16: The modified Artifacts saving directory

```
...
{%- if autosnapshot %}
   _kale_output_artifacts.update({'mlpipeline-ui-metadata': '/tmp/
      mlpipeline-ui-metadata.json'})
   {%- endif %}
   {%- if step.metrics %}
   _kale_output_artifacts.update({'mlpipeline-metrics': '/tmp/
      mlpipeline-metrics.json'})
   {%- endif %}
   {%- if pipeline.processor.id == "nb" and step.name != "
      final_auto_snapshot" and step.name != "pipeline_metrics" %}
   _kale_output_artifacts.update({'mlpipeline-ui-metadata': '/tmp/
      mlpipeline-ui-metadata.json'})
```

```
  _kale_output_artifacts.update({'{{ step.name }}': '/tmp/{{ step.name
      }}.html'})
  {%- endif %}
  {%- if pipeline.processor.id == "py" and step.artifacts and step.
     name != "final_auto_snapshot" and step.name != "pipeline_metrics
     " %}
  _kale_output_artifacts.update({'mlpipeline-ui-metadata': '/tmp/
     mlpipeline-ui-metadata.json'})
  {%- for artifact in step.artifacts %}
  _kale_output_artifacts.update({'{{ artifact["name"] }}': '/tmp' + '
     {{ artifact["path"] }}'})
  {%- endfor %}
  {%- endif %}
...
```

In conclusion, Kale is an admirably useful open-source tool developed by Arrikto. Nevertheless, the security design is missing. Without the described modifications, it wouldn't be possible to utilize Kale in an enterprise environment.

5.3 Deploying KServe on an Enterprise Cluster

The deployment of KServe in a local environment can be achieved effortlessly by applying the relevant customize manifests (Appendix A) and without any further adjustments. Nevertheless, similar to the setup of the tools described in 5.1 and 5.2, installing KServe in the secure enterprise environment of Mercedes-Benz demanded modifications. This section reports the obstacles that occurred during the setup process, the adjustments implemented, and finally, the development of additional features to enhance the tool's usability.

5.3.1 KServe Version and Kubernetes Manifests

The KServe installation in the DnA platform got initiated after the final deployment of Kubeflow Pipelines. The version got selected from the component compatibility matrix

provided by the Kubeflow organization. It is worth mentioning that KServe, is a rebrand of the tool called KFserving. More specifically, in all the releases before the most updated release, v0.7.0 (January 2022), the tool was named as KFServing. However, KServe version 0.7.0 could not be integrated (Figure 5.9) with the rest of the tools deployed. As a result, the most updated supported version got selected: v0.6.1. The KServe installation relies on three more tools: *Istio, Cert Manager, and Knative*. The first two were already present in the DHC CaaS cluster due to their dependencies with the KFP installation. Therefore, the KServe installation in the Mercedes-Benz environment can be divided into two distinct parts: the deployment of the Knative modules and the deployment of KServe. Following is presented an overall outline of each module's utility in serving ML models:

Kubeflow Components

Component	Version	Required
Training Operator	1.4.0	
MPI Operator	1.4.0	
Notebook Controller	1.4.0	
Tensorboard Controller	1.4.0	
Central Dashboard	1.4.0	✓
Profile + KFAM	1.4.0	✓
PodDefaults Webhook	1.4.0	✓
Jupyter Web App	1.4.0	
Volumes Web App	1.4.0	
Katib	0.12.0	
KServe	0.6.1	✓
Kubeflow Pipelines	1.7.0	✓
Kubeflow Tekton Pipelines	1.0.0	

Common Components

Component	Version	Required
Cert-Manager	1.3.1	✓
Istio	1.9.6	✓
Knative	0.22.1	✓

Figure 5.9: The complete list of the Kubeflow Components deployed

1. Knative Components

   - *Knative Serving* is the main Knative module, built on Kubernetes, to support the rapid deployment and serving of applications as serverless containers by providing a simpler deployment syntax. It enables automatic scaling, network, and routing programming. Furthermore, this module is used to expose services via a URL.
   - *Knative Eventing* supports the event-driven characteristic of serverless applica-

tions, and for this setup, it provides inference request logging. Knative Eventing, in particular, is used to transmit and receive events between event producers and sinks using simple POST requests. These events follow the CloudEvents requirements for abstractly expressing event data produced by several sources (e.g., Kafka, S3, GCP PubSub, MQTT).

2. KServe Components

   - *KFServing Controller Manager* is the main component of KServe, used to create the requested services, ingress resources, and the Docker containers for the model servers. Additionally, this module is responsible for generating the model agent container used for request/response logging, batching, and model pulling.
   - *KFServing Models Web App* provides users a UI (Figure 5.10) to manage their Model servers and delivers information derived from the Knative resources, such as live logs from the served model pod. More specifically, using the Models UI, one can:
     - Access the list of the deployed InferenceServices in the user's and contributed namespaces.
     - Create a new InferenceService in the user namespace or a contribution namespace using a YAML file.
     - Delete an existing in the user namespace or a contribution namespace InferenceService.
     - Inspect an InferenceService's status, specs, and YAML definition.
     - Get live logs from the Model server pod.
     - Access metrics when the deployment is integrated with Grafana.

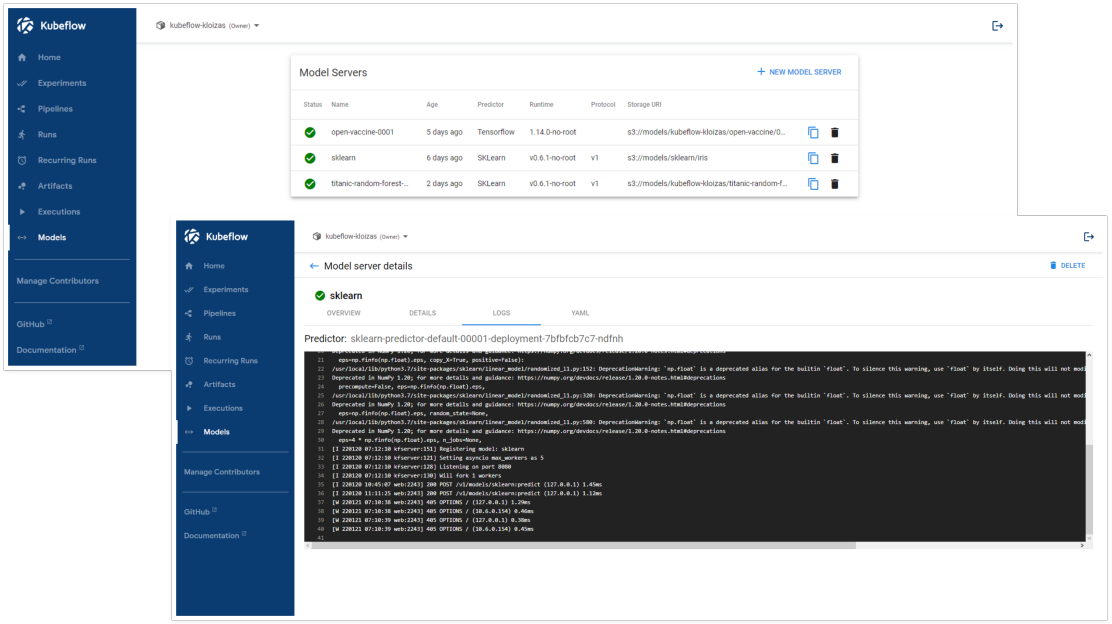

Figure 5.10: Examples of the KServe Models UI

5.3.2 Public Registry Docker Images

Analogous to the case of the Kubeflow Pipeline components, described in 5.1, the modules of KServe, are using Docker images stored in several public container registries such as Docker Hub to make them available to the open-source community. However, Mercedes-Benz utilizes a private Docker registry to monitor which images are used internally and configure them furtherly by adding features such as logging, etc. Since KServe is providing support for all of the popular ML frameworks, the tool's deployment includes:

- The Docker images used for its components
- Several Docker images to enable the usage of ML frameworks such as Tensorflow or Sklearn etc.

All of the Docker images referred to the KServe manifests were pulled, analyzed, optimized to avoid security risks, and pushed to the private Docker registry of Mercedes-Benz, Harbor. The list with the Docker images is included in Figure 5.11.

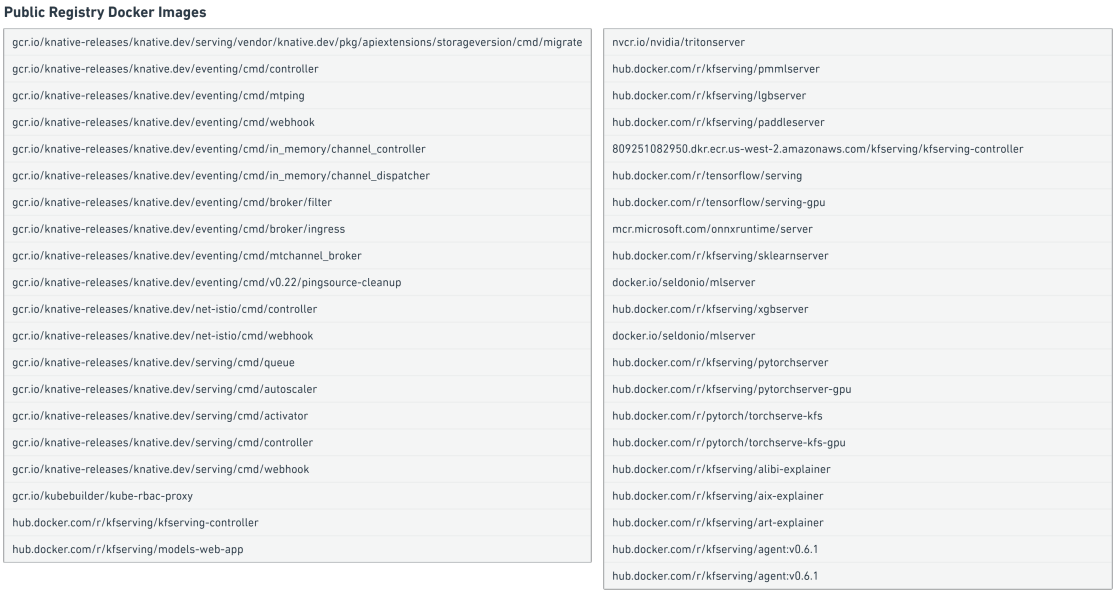

Figure 5.11: The public images used in this deployment

5.3.3 Non-root Installation

The development of this thesis includes an investigation about how enterprise-ready are existing open-source tools. Homogenous to the before-mentioned tools and frameworks, KServe lacks security design. That is due to the default request by most of the docker images used in the deployment for root privileges. The security policy of Mercedes-Benz, as already described in 5.1, is adequately strict and by default prevents the functionality of any Docker image requesting advanced permissions. To overcome this challenge and design a secure version of KSserve, settings similar to the KFP setup were applied. The Security Context options provided by Kubernetes were utilized: *runAsNonRoot*, *fsGroup*, *runAsGroup*, and *runAsUser*. The settings were enforced to all the manifests that were requesting root privileges. Different UIDs and GIDs were selected based on the default images' users..

In conclusion, the conversion of the KServe setup into a secure deployment was demanding. Based on the gained experience from the previous tools installation, the process was completed in a relatively shorter time. Nonetheless, the lack of security orientation also in this tool provides information about the research question indicating that this version of KServe, despite the promising core logic, couldn't be utilized by any enterprise organization applying security policies.

5.3.4 Integration with the DnA Platform

The KServe integration with the DnA platform was initially accomplished by enabling the Models UI in the Kubeflow Central Dashboard. From there, users can access the tool's frontend component, and at the same time, all the Kubeflow tools are in one place. KServe was selected to enable the serving and monitoring part of the MLOps lifecycle. On the one hand, creating an InferenceService is possible via the Models UI. More accurately, users can instantiate model servers, for already stored models, by applying a simple yaml manifest (e.g., Listing 5.17). One of the most crucial features of the tool is that it supports serving models stored in popular storage providers: Google Storage, S3 solutions, Azure Blog Storage, Local Filesystem, and Persistent Volume Claim. In parallel, KServe offers a Python SDK, which includes several functionalities such as registering and serving a stored model, prediction handling, pre/post handling, liveness handling, and readiness handling. While the SDK usage can be convenient, importing the required libraries and referring to the corresponding functions can be challenging and confusing. Hence, the simple installation of the KServe SDK in the Jupyter Notebooks environment was not considered enough since the goal of this project was to create an MLOps workflow to reduce the technical debt and enhance the Data Scientists user experience. The requirement was to furtherly automate the serving process similarly to the Kale and Kubeflow Pipelines one by requiring the minimum effort from the users.

Listing 5.17: Example of an InferenceService yaml file

```
apiVersion: serving.kubeflow.org/v1alpha2
kind: InferenceService
metadata:
  annotations:
    sidecar.istio.io/inject: 'false'
  name: open-vaccine-0013
spec:
  default:
    predictor:
      serviceAccountName: minio-models
      tensorflow:
```

```
    storageUri: s3://models/kubeflow-kloizas/open-vaccine/0013/model.
       tfkeras
minReplicas: 0
```

Kale, the tool used for the automatic conversion of Jupyter Notebooks to Kubeflow pipelines, comes with the KServe Python SDK installed and offers a function for easy integration between the different Kubeflow modules. In particular, Kale provides a utility that recognizes the types of ML models inside a notebook and then uses Rok, a licensed product by Arrikto, to snapshot the current environment and create an InferenceService just by passing a model object. As expected, this solution could not be utilized by the DnA platform because the project's objective required the creation of a holistically open-source MLOps architecture. Nevertheless, due to the open-source format of Kale, modifications were implemented in the tool's source code to develop two new features based on the core integration functionality between the Kale and KServe. In particular, the custom advanced properties were included in the Kale class: *serveutils*.

The first task was to enable serving an already stored model directly from the Jupyter Notebooks. That incorporated the creation of two functions named *serve_from_uri* (Listing 5.18) and *create_inference_service_from_uri* (Listing 5.19).

Listing 5.18: The function developed to serve a stored ML model

```
def serve_from_uri(model_uri: str,
                   predictor: str,
                   name: str = None,
                   wait: bool = True,
                   preprocessing_fn: Callable = None,
                   preprocessing_assets: Dict = None) -> KFServer:

    log.info("Starting serve procedure for model '%s'", model_uri)

    if predictor not in PREDICTORS:
        raise ValueError("Invalid predictor: %s. Choose one of %s"
                         % (predictor, PREDICTORS))
```

```
if not name:
    name = "%s-%s" % (podutils.get_pod_name(), utils.random_string(5)
        )

# Validate and process transformer
if preprocessing_fn:
    _prepare_transformer_assets(preprocessing_fn,
        preprocessing_assets)

log.info("Creating inference service")

kfserver = create_inference_service_from_uri(
    name=name,
    predictor=predictor,
    model_path=model_uri,
    transformer=preprocessing_fn is not None)

if wait:
    monitor_inference_service(kfserver.name)
return kfserver
```

The *serve_from_uri* function is the entry point for the automation of the serving process. More specifically, this is the only function that users are required to reference inside their Jupyter Notebooks, by only passing a minimum set of information about their stored models:

- The link to the model's URI (an S3 bucket for the case of the DnA platform).
- The type of the model's predictor (e.g., *TensorFlow*, *sklearn* or a custom, etc.).
- A name for their model server.

Additionally, users can specify extra parameters as seen in the function definition in Listing 5.18. Then, this function calls the *create_inference_service_from_uri* and passes the information given by users. Based on these data, a new yaml file, the definition for the InferenceService, is created and then applied to the DHC CaaS cluster, utilizing the

default provided Kale functions. As a result, users can directly serve using two code lines a stored model from Jupyter Notebooks, just by importing the *serve_from_uri* and running it.

Listing 5.19: The function developed to create an InferenceService for a stored ML model

```
def create_inference_service_from_uri(name: str,
                                      predictor: str,
                                      model_path: str,
                                      image: str = None,
                                      port: int = None,
                                      transformer: bool = False,
                                      submit: bool = True) -> KFServer:

    if predictor not in PREDICTORS:
        raise ValueError("Invalid predictor: %s. Choose one of %s"
                         % (predictor, PREDICTORS))

    if predictor == "custom":
        if not image:
            raise ValueError("You must specify an image when using a
                custom"
                             " predictor.")
        if not port:
            raise ValueError("You must specify a port when using a custom"
                             " predictor.")
        predictor_spec = CUSTOM_PREDICTOR_TEMPLATE.format(
            image=image,
            port=port,
            model_path=model_path)
    else:
        if image is not None:
            log.info("Creating an InferenceService with predictor '%s'."
                     " Ignoring image...", predictor)
        if port is not None:
            log.info("Creating an InferenceService with predictor '%s'."
```

```
                " Ignoring port...", predictor)
    predictor_spec = MINIO_PREDICTOR_TEMPLATE.format(predictor=
        predictor,
                                            model_path=model_path)

infs_spec = yaml.safe_load(RAW_TEMPLATE.format(name=name))
predictor_spec = yaml.safe_load(predictor_spec)
infs_spec["spec"]["default"]["predictor"] = predictor_spec

if transformer:
    transformer_spec = yaml.safe_load(
        TRANSFORMER_CUSTOM_TEMPLATE.format(
            image=podutils.get_docker_base_image(),
            model_path=model_path
        ))
    infs_spec["spec"]["default"]["transformer"] = transformer_spec

yaml_filename = "/tmp/%s.kfserving.yaml" % name
yaml_contents = yaml.dump(infs_spec)
log.info("Saving InferenceService definition at '%s'", yaml_filename
    )
with open(yaml_filename, "w") as yaml_file:
    yaml_file.write(yaml_contents)

if submit:
    _submit_inference_service(infs_spec, podutils.get_namespace())
return KFServer(name=name, spec=yaml_contents)
```

The second added feature enables serving a model developed in Jupyter Notebooks directly from the working environment. This task was more complex, as an ML model requires before being served to be saved. The solution to this challenge was to exploit the MinIO S3 storage installation provided by the KFP deployment (5.1.1). There, it was created a distinct bucket named *models*. Furthermore, the *serve* function developed by Kale in the *serveutils* was modified (Listing 5.20) to support storing the ML models in the

MinIO bucket, creating an InferenceService based on that, and removing the dependencies to the Rok licensed product. It is also worth mentioning that different model types are saved in distinct formats. For instance, a sklearn model includes only a file, whereas a TensorFlow model comprises a directory with several files. Since Kale is already providing a way to save models locally, the following modifications were developed to achieve the overall goal:

- The addition of the version parameter to secure that different model versions can be stored and served to avoid the application of an InferenceService that has the same name as an existing one.
- The detection of the model's format (a file or a directory) based on which the proper function is called to store the model in MinIO.
- The addition of a function called *save_file_to_minio* (Listing 5.20) where the MinIO Python SDK is utilized, the MinIO client is initialized, and the model file is saved to the model bucket following this template:

  models/user-namespace/model-name/version/model-file
- The addition of a function called *save_dir_to_minio* (Listing 5.20) to store models in MinIO that are stored to the model bucket following this template:

  models/user-namespace/model-name/version/model-directory/model-files
- The deletion of the locally saved models after storing them in MinIO to preserve memory.
- The retrievement of the model's MinIO URI and the call to the *serve_from_uri* function to serve the model.

Listing 5.20: The functions developed to store and serve a ML model from Jupyter Notebooks

```
def serve(model: Any,
          name: str,
          version: str = "latest",
          wait: bool = True,
          predictor: str = None,
          preprocessing_fn: Callable = None,
```

```
     preprocessing_assets: Dict = None) -> KFServer:

log.info("Starting serve procedure for model '%s'", model)
if not version:
   log.info("No version provided, using 'latest'")

# Validate and process transformer
if preprocessing_fn:
   _prepare_transformer_assets(preprocessing_fn,
      preprocessing_assets)

# Detect predictor type
predictor_type = marshal.get_backend(model).predictor_type
if predictor and predictor != predictor_type:
   raise RuntimeError("Trying to create an InferenceService with"
                  " predictor of type '%s' but the model is of type"
                  " '%s'" % (predictor, predictor_type))
if not predictor_type:
   log.error("Kale does not yet support serving objects with '%s'"
           " backend.\n\nPlease help us improve Kale by opening a
              new"
           " issue at:\n"
           "https://github.com/kubeflow-kale/kale/issues",
           marshal.get_backend(model).display_name)
   utils.graceful_exit(-1)
predictor = predictor_type # in case `predictor` is None

# Dump the model
marshal.set_data_dir(PREDICTOR_MODEL_DIR)
model_filepath = marshal.save(model, "model")

model_filename = model_filepath.split('/')[-1]

# Save the model to minio bucket
minio_path = f"{podutils.get_namespace()}/{name}/{version}/{
```

```
        model_filename}"

    model_uri = f"{MINIO_BUCKET}/{podutils.get_namespace()}/{name}/{
        version}"

    model_name = f"{name}-{version}"

    if is_file(model_filepath):
        save_file_to_minio(minio_path, model_filepath, model_uri)
    else:
        save_dir_to_minio(minio_path, model_filepath, model_uri)
        model_parent_dir=minio_path.split('/')[3]
        model_uri= f"{model_uri}/{model_parent_dir}"

    return serve_from_uri(model_uri, predictor, model_name, wait,
        preprocessing_fn, preprocessing_assets)


def save_file_to_minio(minio_path, model_filepath, model_uri):
    minio_client = Minio('minio-service.kubeflow.svc.cluster.local:9000'
        , access_key='*****', secret_key='****', secure=False)
    try:
        minio_client.fput_object(MINIO_BUCKET, minio_path, model_filepath
            )
        log.info("Model file saved successfully at {}".format(model_uri))
        # delete the locally saved model file
        os.remove(model_filepath)


    except InvalidResponseError as err:
        raise InvalidResponseError("Invalid response error {}".format(err
            ))


def save_dir_to_minio(minio_path, model_filepath, model_uri):
```

```
for model_file in glob.glob(model_filepath + '/**'):
    if os.path.isdir(model_file):
        save_dir_to_minio(
            minio_path + "/" + os.path.basename(model_file), model_file
                , model_uri)
    else:
        remote_path = os.path.join(
            minio_path, model_file[1 + len(model_filepath):])
        save_file_to_minio(remote_path, model_file, model_uri)
```

As a result, the adjusted *serve* function operates as the entry point for serving models developed in Jupyter Notebooks directly from the same workplace. The original utility included in Kale provides the detection of the model's predictor type. That is to say that after the implemented modifications, the DnA users can serve their ML models just by importing the serve function and passing a minimum set of information:

- the model object they want to serve
- the name of the InferenceService
- the version of the InferenceService

Additionally, users can specify extra parameters as seen in the function definition in Listing 5.20.

The implemented features contribute to the enhancement of the Data user experience. The DnA platform's MLOps solution offers three easy ways to serve a model:

1. Via the Models UI to serve a stored model by a simple yaml definition.
2. Via Jupyter notebooks to serve a stored model by a one-line function.
3. Via Jupyter notebooks to serve a developed model directly from the workspace using a one-line function.

Finally, getting predictions from the served models is an easy process. It can be achieved by simple POST requests in the model predict-URL provided by KServe. POST requests can be performed either by a CLI or directly from the Jupyter Notebooks (Figure 5.12.

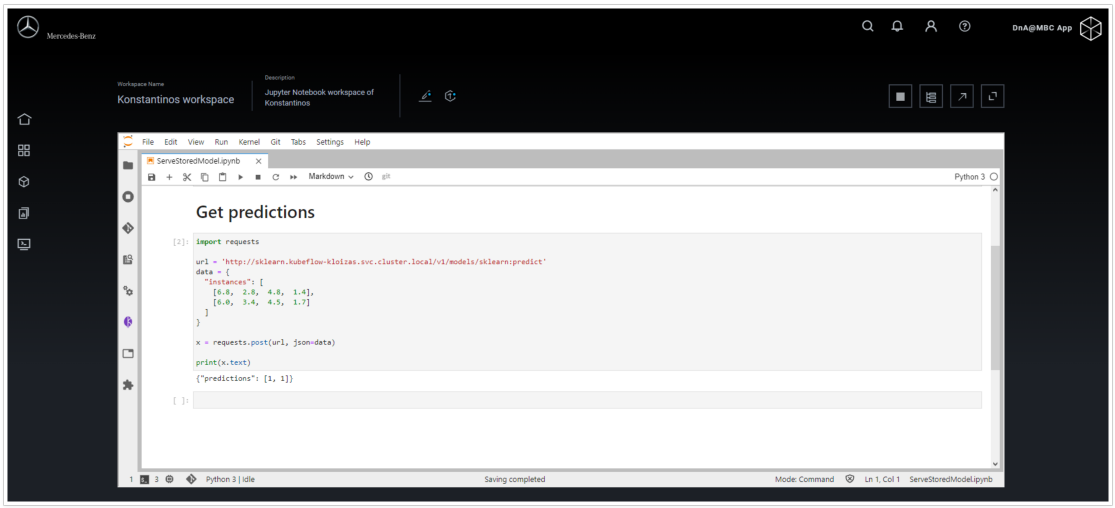

Figure 5.12: Getting predictions with a simple POST request

In conclusion, KServe is a handy tool provided by the Kubeflow ecosystem to enable serving models abstractly via the Models UI or the Python SDK. Nonetheless, the tool's installation in the secure enterprise environment couldn't be accomplished without the described modifications.

5.4 The Answer to the Research Question

In addition to the MLOPs lifecycle implementation, of equal importance for the current thesis was the exploration task enterprise-readiness of existing FOSS solutions (Section 2.2). Undeniably, the sample of the open-source software used in this project was significantly small. However, the Kubeflow ecosystem is a project that originates its initial development to the Google team. Furthermore, it is a project that aims to constitute the primary choice for organizations performing Data and Analytics operations. Despite that fact, the hands-on experience gained by deploying the tools to the environment of the Mercedes-Benz AG proved multiple times that the FOSS Kubeflow is not ready for the enterprise world. More specifically, the installation process of every tool was not straightforward at all and required, in most cases, challenging modifications. Most of the project effort and time was spent on adjusting the numerous components, more than 30, to secure versions. Software Security is probably the most crucial factor for organizations around the world. Additionally, Security is one of the main aspects of the ISO/IEC 25010

(2011) Software Quality Model, which is used to evaluate the quality of software (Figure 5.13). In this case, the conclusions extracted by the project lead to the deduction that the quality of existing FOSS solutions, such as Kubeflow, is dramatically affected by the lack of security design. Although ISO/IEC 25010 (2011) characteristics such as Performance Efficiency can be found in many open source solutions, the lack of Security properties such as Confidentiality or Integrity usually results in the ostracism of FOSS from the enterprise environments.

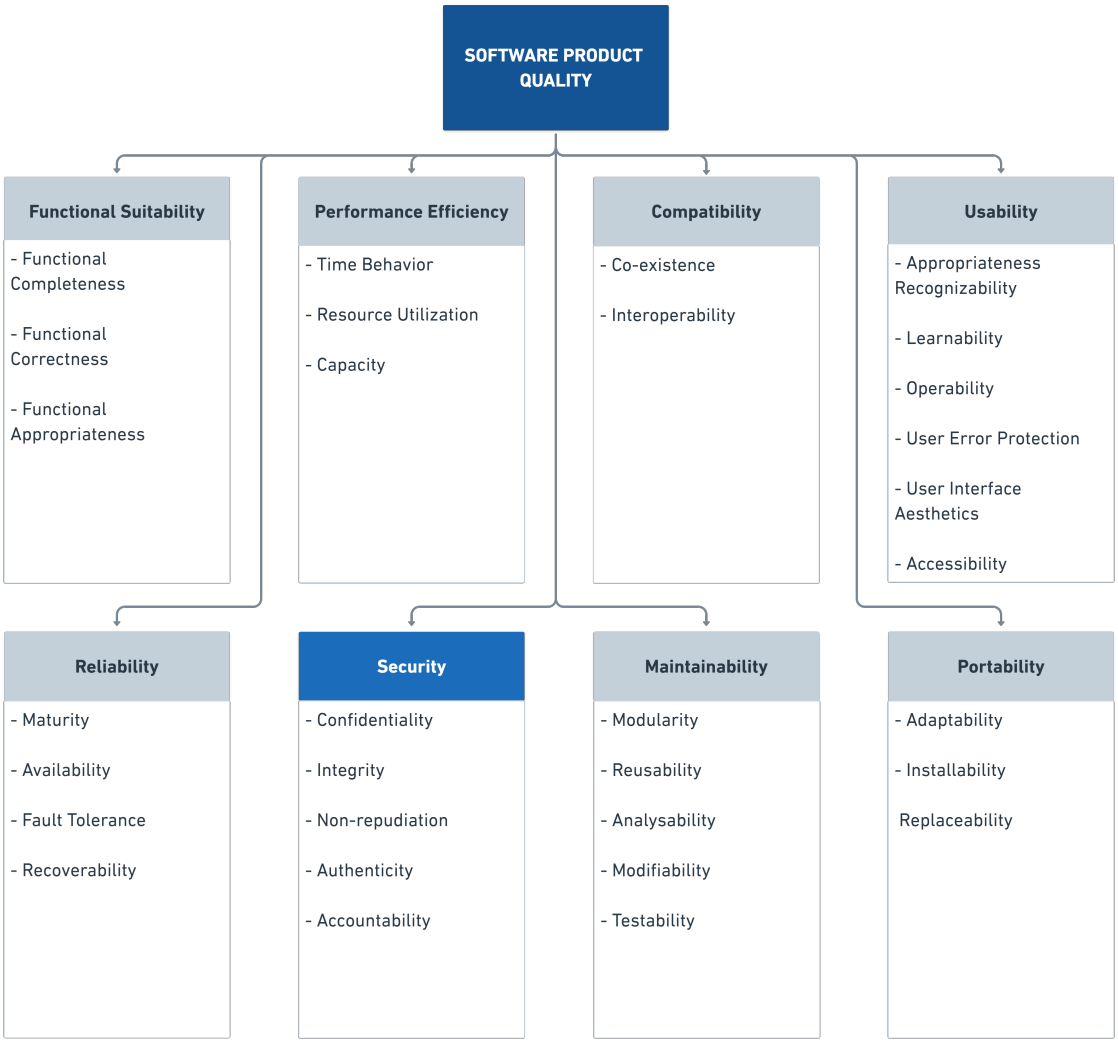

Figure 5.13: The Software Product Quality Model (ISO/IEC 25010, 2011)

6 Evaluation

During the implementation process of the current project, several open-source ML notebooks were used to trial the system's functionality. The utilization of such models was indubitably necessary and enabled an efficient and effective way of spotting and eliminating failures and bugs. Initially, the final form of the developed MLOps lifecycle was successfully tested using the forenamed models. However, the solution validation within the enterprise environment of Mercedes-Benz AG was realized utilizing an existing ML model, Chronos. This chapter presents the automation of the Chronos model development using the DnA platform and the designed MLOps framework and provides a comprehensive overview of the developed features.

6.1 The Chronos Use Case

The Chronos team within Mercedes-Benz AG is developing a time-series ML model to enable forecasting predictions using data stamped on historical time. The publication of its source code is not allowed since this is an internal company project. Nevertheless, the developing team provided one of their core Notebooks with dummy data and configuration files to validate the solution of the thesis. It is worth mentioning that the Chronos model was developed in a licensed product environment, so the confirmation of the solution functionally could initiate the process of moving the project into the open-source space. The Chronos Notebook is a typical form of an ML notebook. It contains several cells where different actions such as Imports, Data Reading, Exploratory Data Analysis, Model Training, etc., are taking place (Figure 6.1).

The first step of the automation includes the conversion of the notebook into a KF

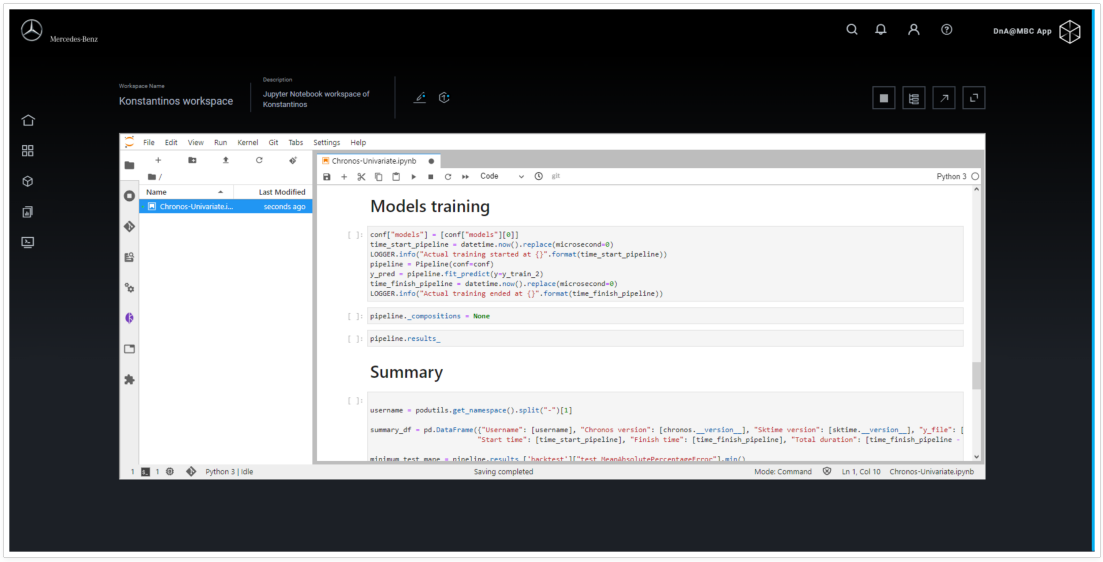

Figure 6.1: A sample of the Chronos model

pipeline. That is accomplished by utilizing Kale. Users should enable the tool from the Jupyter UI and annotate their notebooks. In this case, the Chronos notebook annotations were based on the desired execution steps. In parallel to the annotation of the cells, the dependencies between the different pipeline stages were declared (Figure 6.2).

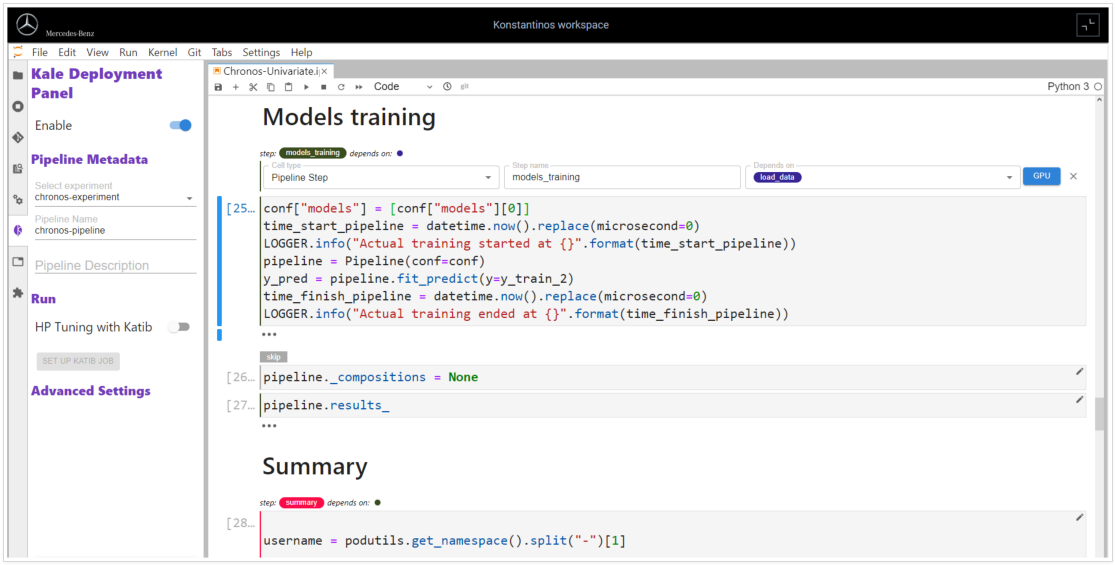

Figure 6.2: Creating annotations for the different pipeline steps

Next, the pipeline was assigned to an experiment and named accordingly via the Kale UI. The last step includes clicking the "Compile and Run" button to automatically create the KF pipeline, upload it to KFP and run it (Figure 6.3). To conclude, these two

simple steps enable users to move their Jupyter Notebooks in KFP without any further technical knowledge required.

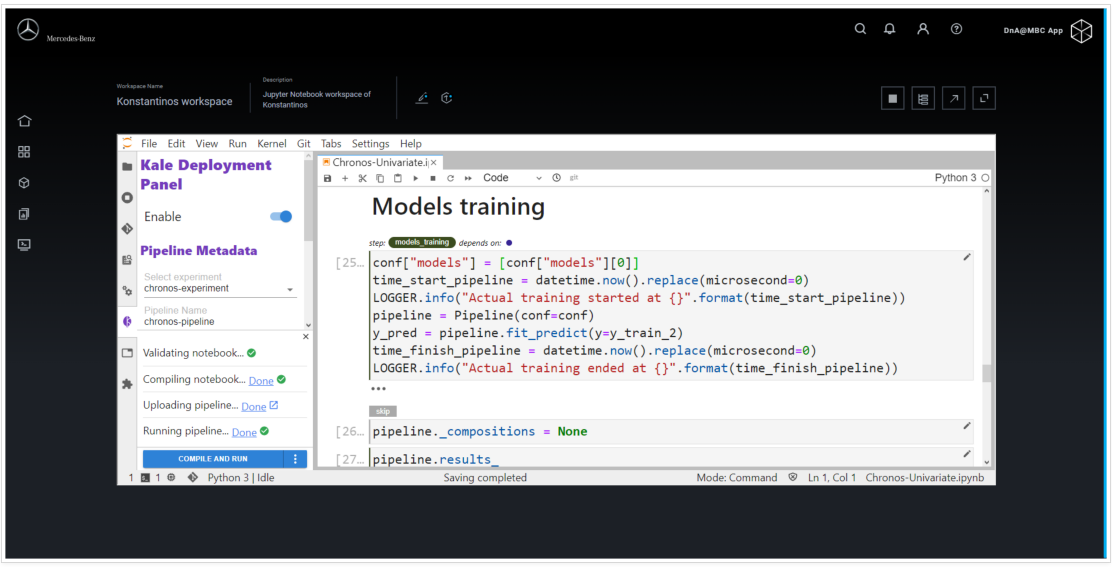

Figure 6.3: Compiling, uploading and running the generated KF pipeline

Figure 6.4 contains the result of the executed pipeline. One can already spot some of the handful KFP features. The most important one includes the parallel execution of independent pipeline steps (EDA and Model Training). This feature enables users to run their KF pipelines faster and more efficiently.

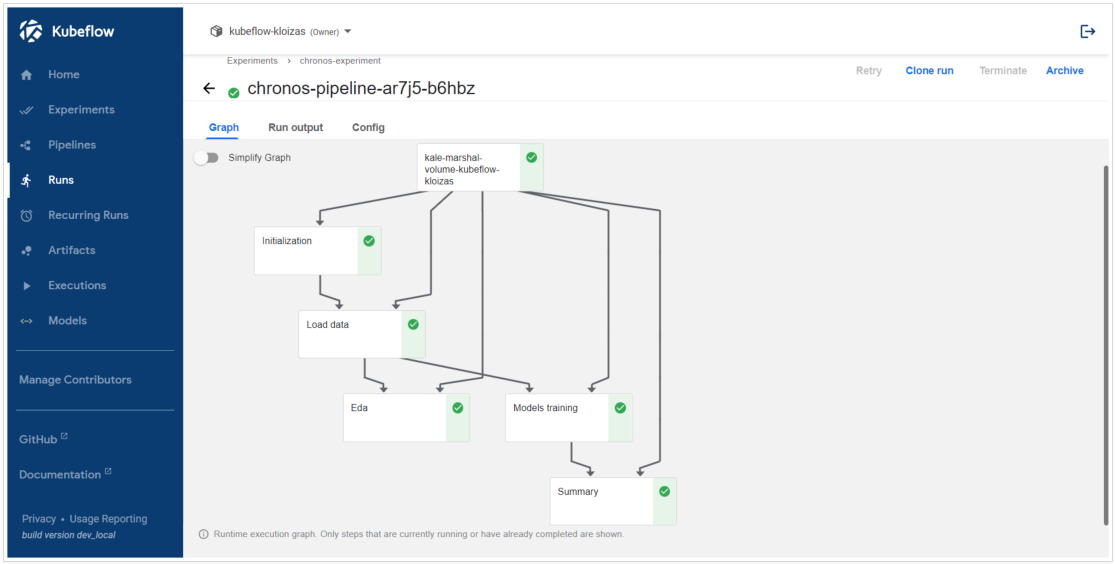

Figure 6.4: The successfully executed Chronos KF pipeline

Figure 6.5 presents an example of cached execution. More specifically, the green

symbol in each step indicates that the particular stage was not executed. The results are loaded from the cache server because the steps already ran in the past without any change. That is a particularly convenient feature. Only the modified pipeline steps and their interdependencies have to be re-executed again. Thus, time and resource costs are highly impacted and get significantly reduced.

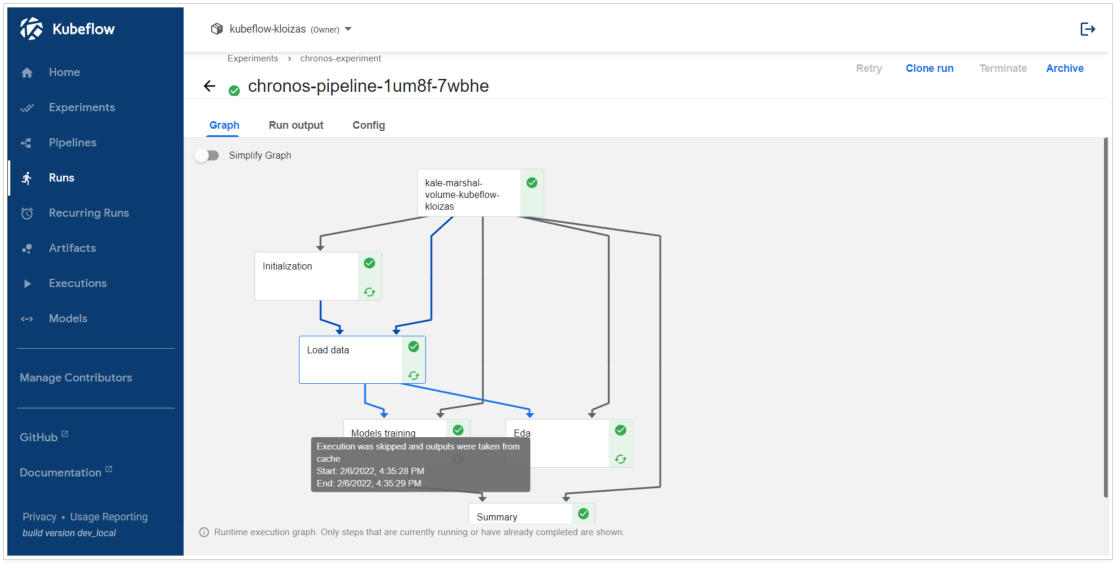

Figure 6.5: A caching example

The Central Dashboard UI provides several features. For instance, users can access handy information about the execution and get beautiful visualizations by clicking on individual pipeline steps (Figure 6.6).

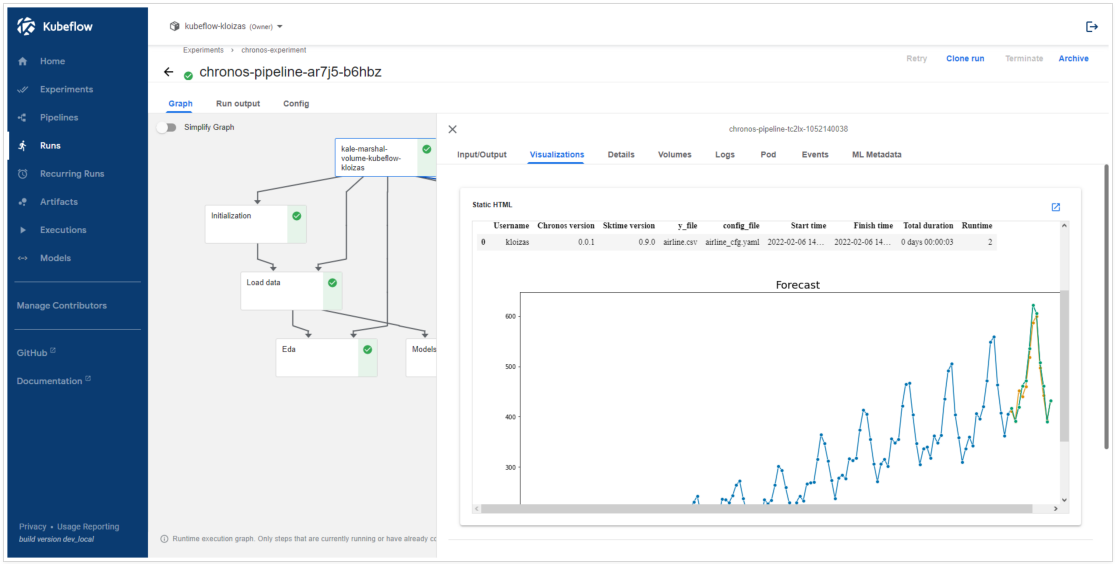

Figure 6.6: A visualization example from a KFP step

Additionally, Experiments enable efficiently organizing the different pipeline versions. Users can select multiple pipelines and compare them using metrics (Figure 6.7).

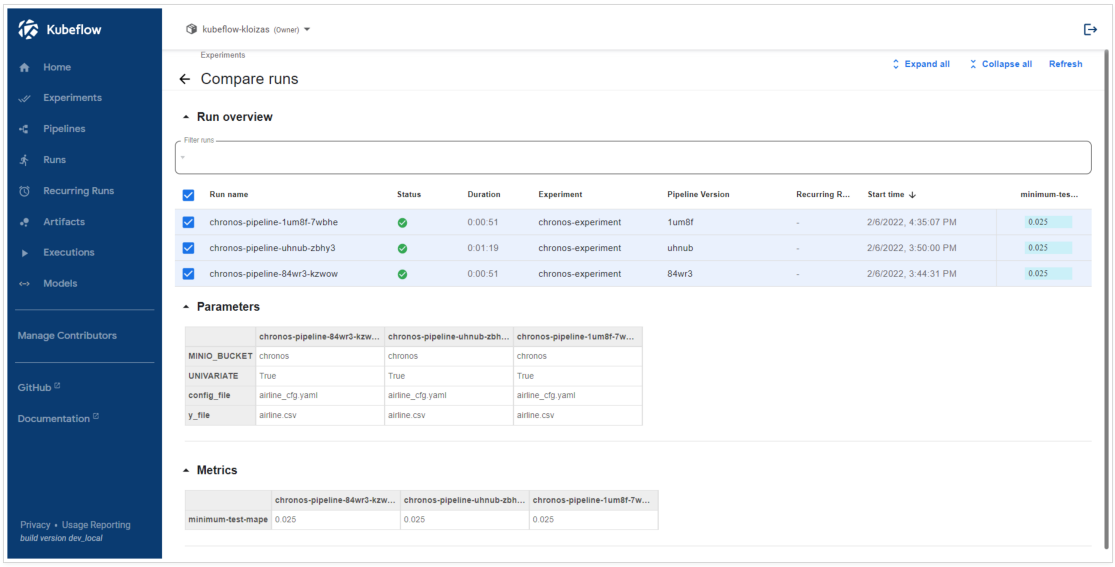

Figure 6.7: Comparing different KF pipeline versions

The next step of the MLOps lifecycle is the model serving. That can be achieved in three different ways described in Chapter 5.3. However, serving time series forecasting models is not a common practice. Nonetheless, a model server was developed for the Chronos use case to demonstrate the feature of custom ML predictors. Usually, the process of serving regular model types such as a TensorFlow or Pytorch solution within the implemented MLOps pipeline requires only the declaration of the model, its name, and its version. For the Chronos case, two extra parameters were essential: the statement of the custom predictor type and the reference to the implemented model server. (Figure 6.8). The model is served effortlessly, just in one line of code. A hyperlink leads to the Models UI in the Kubeflow environment.

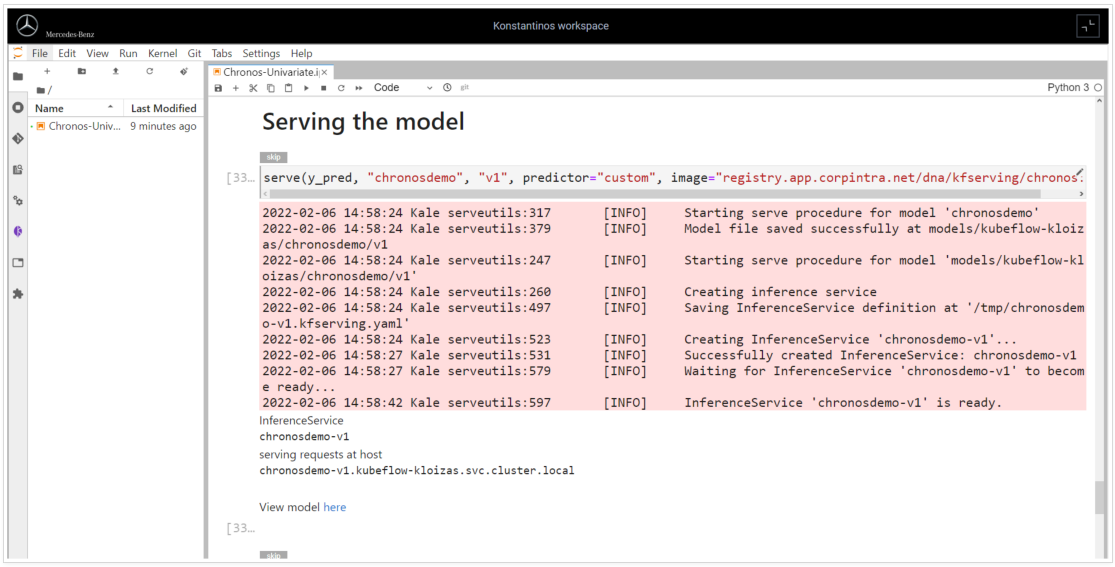

Figure 6.8: Serving a custom model from Jupyter Notebooks

Using the Models UI, users get an overview of their served models and access crucial information about each one of them (Figure 6.9). Predictions, as described in Chapter 5.3, can be made with a simple HTTP request.

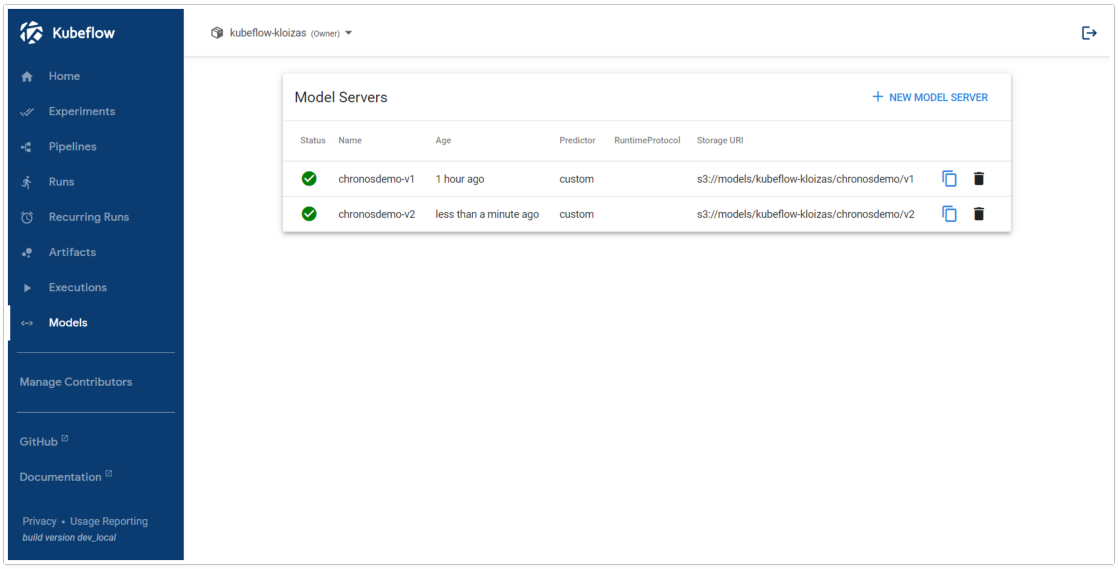

Figure 6.9: The Models UI

Finally, it should be mentioned that several additional features are included in the Central Dashboard UI. Users can trigger or schedule KF pipeline runs, access pipeline artifacts, and more. One of the most crucial features enables users to provide access to their namespace to others. That can be done straightforwardly via the Contributors tab by simply declaring the user's short id (Figure 6.10).

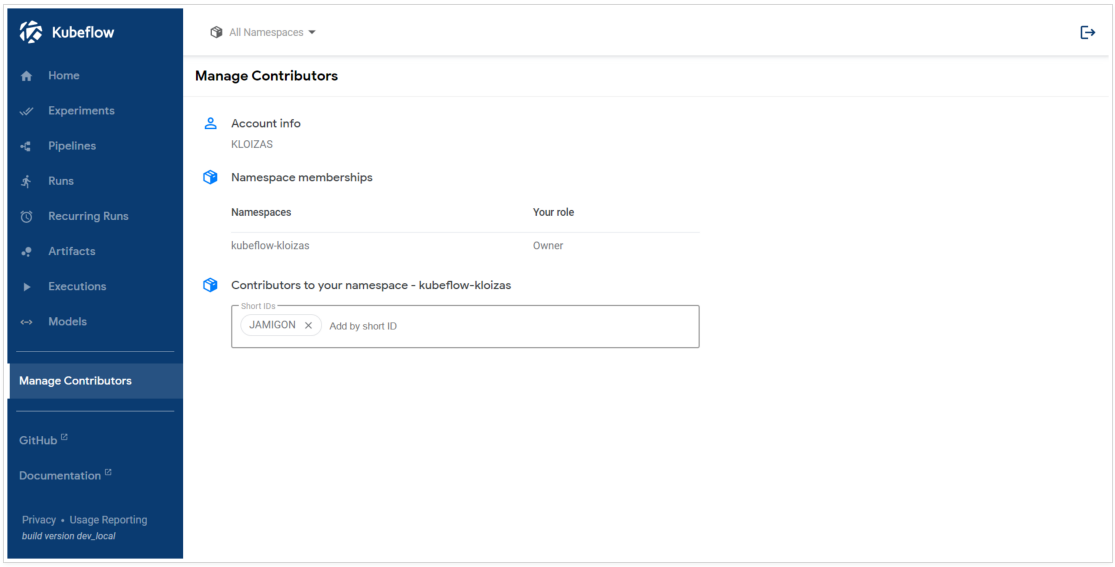

Figure 6.10: Adding contributors

In conclusion, the implemented solution was successfully validated by the Chronos model use case. The benefits of the designed MLOps lifecycle can enable cost, time, and effort reductions. The result of this demonstration was the initiation of discussions and plans for the future movement of the Chronos development to the open-source environment of the DnA platform. That fact confirms the practical value of the implemented project and indicates the generic benefits of utilizing FOSS tools in enterprise environments.

7 Conclusions and Outlook

While AI and ML development are becoming the defacto standard in everyday enterprise life, ML solutions are still struggling to find their way to production. A high percentage of the developed within organizations ML systems is kept to laboratory environments as quite often Data Scientists do not have the required technical skills to scale, deploy and distribute their solutions. The outcome is the creation of technical debt, mostly between Data Scientists and Software engineers, and eventually, the increment in resources costs for organizations. MLOps is a new and rapidly rising framework based on DevOps principles and aims to tackle this challenge by automating the steps of creating machine learning systems. In 2022 there are various MLOps tools, many of which are open-source projects. The importance of the FOSS for the global development community is so vital that nowadays, almost every organization is making use of open software. FOSS can only benefit enterprises in many ways, such as enabling software development teams to focus on the core product idea by utilizing open-source components instead of building everything from scratch. In return, organizations using open software tend to contribute back to FOSS communities by either enhancing the functionality of existing projects or opening their products' source code. Mercedes-Benz AG relies on open-source components for many of its projects. One of the most important company's internal projects in the Data Science field, the Data and Analytics platform, constitutes the first significant open-source contribution of Mercedes-Benz. Essentially, the DnA platform is a complete solution in the Analytics area that utilizes open-source technologies to enable users to access data, create, manage and share ML solutions conveniently and transparently.

The task of this thesis was to design, implement a complete MLOps lifecycle and integrate it with the DnA platform, using existing open-source tools while in parallel

investigating and documenting whether FOSS solutions are enterprise-ready. In particular, the main objective was to enable Data Scientists to transfer their ML solutions from their working space, that is, Jupyter Notebooks, to ML pipelines in an as much as possible effortless way. The first step to tackle the problem included the selection of existing open-source solutions. Kubeflow, developed by Google, constitutes one of the most promising FOSS ML toolkits. Consisting of many different tools, Kubeflow is essentially a Cloud-Native framework designed to simplify the running process of ML workflows on top of Kubernetes. For this project, only a fraction of Kubeflow modules were selected. The most important of the components utilized are:

- Kubeflow Pipelines is a tool that enables automatically creating and executing scalable, portable, and ML workflows.
- Kale, an addon tool contributed by Arrikto, is a convenient extension that automatically converts ML solutions written in Jupyter Notebooks to Kubeflow Pipelines in a non-complex way.
- KServe is used to scalably serve and monitor, in an abstract way, ML models that are based on popular or custom ML frameworks.

The deployment of the Kubeflow modules into the secure enterprise environment of Mercedes-Benz was challenging and required plenty of modifications in the different components. The main reason included the request for advanced privileges in the company cluster from most of the docker images used by the Kubeflow components. The overwhelming majority of the tools' modules had to be adjusted to remove the security risks. This particular designed MLOps workflow includes more than 20 different parts. As a result, a significant amount of time and effort was spent to set up the secure versions. The crucial modification included the application of security context settings that would require the deployments to run docker images as non-root users. The fact that most of the setup time, about four of the six project months, was required to convert the tools into their secure versions produces concrete evidence about FOSS. Despite the numerous benefits, open-source solutions often lack security design, and therefore they cant be utilized in enterprise environments. While someone would expect that at least big FOSS projects like Kubeflow from Google would be secure by default, it seems that this

is not the case. Hence, maintainers and potential users end up with additional effort as no root-requiring tool can exist in enterprises. That leads to the conclusion that there are high chances that open-source solutions can very often end up due to their from scratch lack of security design as a Proof of Concept projects, instead of being utilized in real-life scenarios by organizations around the world.

However, the security adjustments led to establishing a modern and multi-user isolated MLOps lifecycle that can practically benefit the users and organizations. Data Scientists can develop their ML models in their already known environment of Jupyter Notebooks. From there, without hardly any advanced technical expertise, they can put their solutions into Kubeflow Pipelines using the Jupyterlab extension, Kale, just by annotating their cells and clicking a button. Furthermore, they can serve their models in an API form, using one-line commands directly from the notebook or the UI. With simple POST requests, they can get predictions from their models' servers via any CLI or even directly from inside the Jupyter Notebooks. Finally, via the handy Central Dashboard UI, authorized users can access their pipelines and their served models to perform a set of different actions. For instance, they can schedule KFP runs, compare metrics between pipeline executions, add contributors to enable other users to access experiments, and more. The solution was applied successfully to the real-life use case scenario of the Chronos model. That acted as a validation of the initially planned objective. The technical debt between Data Scientists and Software Engineers is eliminated. That is achieved via a secure, open-source, multi-user isolated, deployable, and scalable MLOps lifecycle in a transparent, configuration-free, and technical expertise-independent way. ML solutions can now reach production in an easier than ever way.

The project of this thesis became a crucial component of Mercedes-Benz's DnA platform and was included in the Q2 2022 release, published in: `https://github.com/mercedes-benz/DnA/`. On a personal note, I was honored to be part of the DnA platform team in this critical Mercedes-Benz project. I hope that more and more worldwide enterprises will follow this paradigm and will develop secure open-source solutions.

7.1 Future Work

A bi-product of this thesis is the actual hands-on experience of the feature set that the deployed tools and technologies offer. The designed MLOps lifecycle has succeeded in being a sufficiently functional solution. However, several installed components require further improvements to enforce efficient and safe workflow operation. In addition, the overall functionality can be further expanded and optimized by developing or integrating new features into the existing architecture. The following is a list of future recommendations:

1. Extend multi-user isolation to support access control for Pipeline definitions, Executions, and MinIO artifact storage.
2. Enable the selection of work that can be shared contributors, such as specific experiments, etc.
3. Enable granular contributor permissions to give specific permissions to particular contributors.
4. Enable group Kubeflow profiles for teams working on projects.
5. Enable uploading files in the UI to adjust the execution parameters of pipeline runs.
6. Extend the UI by integrating a model registry, such as MinIO, to enable serving, stored models with a click.
7. Enable notification service for KFP and KServe.
8. Upgrade Kale to support the newest JupyterLab version.
9. Enable users to create their custom Jupyter Notebooks environment to include specific library versions.
10. Integrate a data versioning tool such as the open-source DVC.

Finally, it should be pointed out again that FOSS solutions need to be secure to utilize by organizations. Designing secure open-source software from scratch will enable the faster and broader absorption of the projects and will reduce the additional effort of discovering and fixing potential risks.

List of Figures

Listings

References

Alqudah, M., & Razali, R. (2016). A review of scaling agile methods in large software development. *International Journal on Advanced Science, Engineering and Information Technology*, *6*(6), 828. doi: 10.18517/ijaseit.6.6.1374

Alzubi, J., Nayyar, A., & Kumar, A. (2018). Machine learning from theory to algorithms: An overview. *Journal of Physics: Conference Series*, *1142*, 012012. doi: 10.1088/1742-6596/1142/1/012012

Andersen-Gott, M., Ghinea, G., & Bygstad, B. (2012). Why do commercial companies contribute to open source software? *International Journal of Information Management*, *32*(2), 106–117. doi: 10.1016/j.ijinfomgt.2011.10.003

Angra, S., & Ahuja, S. (2017). Machine learning and its applications: A review. *2017 International Conference on Big Data Analytics and Computational Intelligence (ICBDAC)*. doi: 10.1109/icbdaci.2017.8070809

Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Software*, *33*(3), 42–52. doi: 10.1109/ms.2016.64

Bisong, E. O. (2019). *Building machine learning and deep learning models on google cloud platform: A comprehensive guide for beginners*. Apress.

Boehm, B. W. (1987, Sep). Industrial software metrics top 10 list. *IEEE Software*, *4*(5), 84–85.

Bonaccorsi, A., & Rossi, C. (2003). Why open source software can succeed. *SSRN Electronic Journal*. doi: 10.2139/ssrn.348301

Bonaccorsi, A., & Rossi, C. (2006). Comparing motivations of individual programmers and firms to take part in the open source movement: From community to business. *Knowledge, Technology amp; Policy*, *18*(4), 40–64. doi: 10.1007/s12130-006-1003-9

Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., & Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities. *Journal of Internet Services and Applications*, *9*(1). doi: 10.1186/s13174-018-0087-2

Brenner, R., & Wunder, S. (2015). Scaled agile framework: Presentation and real world example. *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. doi: 10.1109/icstw.2015.7107411

Breuel, C. (2020, Jan). *Ml ops: Machine learning as an engineering discipline.* Towards Data Science. Retrieved from https://towardsdatascience.com/ml-ops-machine-learning-as-an-engineering-discipline-b86ca4874a3f

Cekic, B. (2021, Jun). *Here is why azure devops is a must-have tool for agile (sapui5) projects.* SAP Community Blogs. Retrieved from https://blogs.sap.com/2020/12/04/here-is-why-azure-devops-is-a-must-have-tool-for-agile-sapui5-projects/

Chauhan, N. K., & Singh, K. (2018). A review on conventional machine learning vs deep learning. *2018 International Conference on Computing, Power and Communication Technologies (GUCON)*. doi: 10.1109/gucon.2018.8675097

Cois, C. A., Yankel, J., & Connell, A. (2014). Modern devops: Optimizing software development through effective system interactions. *2014 IEEE International Professional Communication Conference (IPCC)*. doi: 10.1109/ipcc.2014.7020388

Columbus, L. (2021, Jan). *76% of enterprises prioritize ai amp; machine learning in 2021 it budgets.* Forbes Magazine. Retrieved from https://www.forbes.com/sites/louiscolumbus/2021/01/17/76-of-enterprises-prioritize-ai--machine-learning-in-2021-it-budgets/

Dahlander, L. (2005). Appropriation and appropriability in open source software. *International Journal of Innovation Management*, *09*(03), 259–285. doi: 10.1142/s1363919605001265

Dahlander, L., & Magnusson, M. (2008). How do firms make use of open source communities? *Long Range Planning*, *41*(6), 629–649. doi: 10.1016/j.lrp.2008.09.003

Daimler. (2020a). *Company history.* Retrieved from https://www.daimler.com/company/tradition/company-history/

Daimler. (2020b, Dec). *Daimler annual report 2020.* Retrieved from https://annualreport.daimler.com/2020/

Daimler. (2020c). *Daimler/daimler-foss: A collection of information on daimler open source stuff - code of conduct, daimler cla, and more.* Retrieved from https://github.com/Daimler/daimler-foss

Das, S. (2021, Sep). *Why jupyter notebooks are so popular among data scientists.* Retrieved from https://analyticsindiamag.com/why-jupyter-notebooks-are-so-popular-among-data-scientists/

Driver, M., & Klinect, T. (2019). *Hype cycle for open-source software.* Retrieved from https://www.gartner.com/en/documents/3955972/hype-cycle-for-open-source-software-2019

Ebert, C. (2007). Open source drives innovation. *IEEE Software*, *24*(3), 105–109. doi: 10.1109/ms.2007.83

Ebert, C. (2008). Open source software in industry. *IEEE Software*, *25*(3), 52–53. doi: 10.1109/ms.2008.67

Ebert, C. (2009). Guest editor's introduction: How open source tools can benefit industry. *IEEE Software*, *26*(2), 50–51. doi: 10.1109/ms.2009.38

El Naqa, I., & Murphy, M. J. (2015). What is machine learning? *Machine Learning in Radiation Oncology*, 3–11. doi: 10.1007/978-3-319-18305-3_1

Fioravanzo, S. (2019, Nov). *Automating jupyter notebook deployments to kubeflow pipelines with kale.* kubeflow. Retrieved from https://medium.com/kubeflow/automating-jupyter-notebook-deployments-to-kubeflow-pipelines-with-kale-a4ede38bea1f

Fioravanzo, S., & Koukis, V. (2020, Feb). Retrieved from https://www.cncf.io/wp-content/uploads/2020/08/CNCF-webinar-February-27th-From-Notebook-to-Kubeflow-Pipelines-with-MiniKF-Kale-1.pdf

Freeman, E. (2019). *What is devops?* Retrieved from https://aws.amazon.com/devops/what-is-devops/

Frikha, A. (2021, Jul). *From notebooks to pipelines with kubeflow kale.* Ubuntu. Retrieved from https://ubuntu.com/blog/kubeflow-kale

FSF. (2021). *Free software foundation (fsf).* Retrieved from https://www.fsf.org/

about/

Goltsman, K. (2019, Mar). *Defining privileges and access control settings for pods and containers in kubernetes.* Supergiant.io. Retrieved from https://medium.com/kubernetes-tutorials/defining-privileges-and-access-control-settings-for-pods-and-containers-in-kubernetes-2cef08fc62b7

Google. (2020). *Mlops: Continuous delivery and automation pipelines in machine learning.* Author. Retrieved from https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

Google. (2021a). *Google trends - mlops.* Author. Retrieved from https://trends.google.com/trends/explore?date=2017-01-01%202021-10-06&q=mlops

Google. (2021b, Aug). *Introduction to kubeflow.* Retrieved from https://www.kubeflow.org/docs/about/kubeflow/

Guerrero, J. (2021, Nov). *Introducing the kale sdk for mlops and kubeflow: Arrikto.* Retrieved from https://www.arrikto.com/blog/introducing-the-kale-sdk-for-mlops-and-kubeflow/

Gürsakal, N., Çelik, S., & Gürsakal, S. (2021). Big data companies and open source movement. *European Journal of Science and Technology*. doi: 10.31590/ejosat.822219

Haghirian, P., & Kayser, C. (2018). Case study 6: Daimler entering new markets. In *Business development, merger and crisis management of international firms in japan: Featuring case studies from fortune 500 companies* (p. 91–106). World Scientific.

Haviv, Y. (2020, Sep). *Productionizing machine learning with a microservices architecture.* Retrieved from https://databricks.com/session_na20/productionizing-machine-learning-with-a-microservices-architecture

ISO/IEC 25010. (2011). *ISO/IEC 25010:2011, systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models.*

Jabbari, R., bin Ali, N., Petersen, K., & Tanveer, B. (2016). What is devops? *Proceedings of the Scientific Workshop Proceedings of XP2016*. doi: 10.1145/2962695.2962707

K, B. (2020, Dec). *Everything you need to know about jupyter notebooks.* Towards Data Sci-

ence. Retrieved from https://towardsdatascience.com/everything-you-need-to-know-about-jupyter-notebooks-10770719952b

Kang, M., & Jameson, N. J. (2018). Machine learning: Fundamentals. *Prognostics and Health Management of Electronics*, 85–109. doi: 10.1002/9781119515326.ch4

Karamitsos, I., Albarhami, S., & Apostolopoulos, C. (2020). Applying devops practices of continuous automation for machine learning. *Information*, *11*(7), 363. doi: 10.3390/info11070363

Katsakioris, I. (2021). *Support of jupyterlab 3.0.1 · issue 282 · kubeflow-kale/kale.* Retrieved from https://github.com/kubeflow-kale/kale/issues/282#issuecomment-762135467

Klettke, M., & Störl, U. (2021). Four generations in data engineering for data science. *Datenbank-Spektrum*. doi: 10.1007/s13222-021-00399-3

Kretz, A. (2019). *The data engineering cookbook*.

KServe. (2021). *Kserve v0.7 is released, read blog.* Retrieved from https://kserve.github.io/website/blog/articles/2021-10-11-KServe-0.7-release/

Kubeflow. (2021a, Nov). *Kubeflow overview.* Retrieved from https://www.kubeflow.org/docs/started/kubeflow-overview/

Kubeflow. (2021b, Nov). *Overview of kubeflow pipelines.* Retrieved from https://www.kubeflow.org/docs/components/pipelines/overview/pipelines-overview/

Kubeflow-authors. (2021, Mar). *Pipelines api reference.* Retrieved from https://www.kubeflow.org/docs/components/pipelines/reference/api/kubeflow-pipeline-api-spec/

Kubeflow-Kale. (2021). *Kale/pipeline_template.jinja2 at v0.7.0.* Retrieved from https://github.com/kubeflow-kale/kale/blob/v0.7.0/backend/kale/templates/pipeline_template.jinja2

Kubernetes-authors. (2021, Nov). *Configure a security context for a pod or container.* Retrieved from https://kubernetes.io/docs/tasks/configure-pod-container/security-context/

Leffingwell, D., Knaster, R., Oren, I., & Jemilo, D. (2018). *Safe reference guide: Scaled agile*

framework for lean enterprises. Scaled Agile, Inc.

Leite, L., Rocha, C., Kon, F., Milojicic, D., & Meirelles, P. (2020). A survey of devops concepts and challenges. *ACM Computing Surveys*, *52*(6), 1–35. doi: 10.1145/3359981

Lwakatare, L. E., Kuvaja, P., & Oivo, M. (2016, Aug). An exploratory study of devops extending the dimensions of devops with practices. *The Eleventh International Conference on Software Engineering Advances*.

Metacontroller. (2021). *Metacontroller.* Retrieved from https://metacontroller.github.io/metacontroller/intro.html

Meulen, R. v. d., & McCal, T. (2018). *Gartner says nearly half of cios are planning to deploy artificial intelligence.* Retrieved from https://www.gartner.com/en/newsroom/press-releases/2018-02-13-gartner-says-nearly-half-of-cios-are-planning-to-deploy-artificial-intelligence

Miller, R. (2022, Jan). *With more data available than ever, are companies making smarter decisions?* TechCrunch. Retrieved from https://techcrunch.com/2022/01/07/with-more-data-available-than-ever-are-companies-making-smarter-decisions

Mitchell, T. M. (1997). *Machine learning*. MacGraw-Hill.

Patel, A. (2018, Jul). *Machine learning algorithm overview.* ML Research Lab. Retrieved from https://medium.com/ml-research-lab/machine-learning-algorithm-overview-5816a2e6303

Patterson, J., Katzenellenbogen, M., & Harris, A. (2021). *Kubeflow operations guide: Managing cloud and on-premise deployment*. O'Reilly Media.

Perera, P., Silva, R., & Perera, I. (2017). Improve software quality through practicing devops. *2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer)*. doi: 10.1109/icter.2017.8257807

Perkel, J. M. (2018). Why jupyter is data scientists' computational notebook of choice. *Nature*, *563*(7729), 145–146. doi: 10.1038/d41586-018-07196-1

Randles, B. M., Pasquetto, I. V., Golshan, M. S., & Borgman, C. L. (2017). Using the jupyter notebook as a tool for open science: An empirical study. *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. doi: 10.1109/jcdl.2017.7991618

Ray, S. (2019). A quick review of machine learning algorithms. *2019 International Confer-*

ence on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon). doi: 10.1109/comitcon.2019.8862451

Reddi, V. J., Diamos, G., Warden, P., Mattson, P., & Kanter, D. (2021, Feb). Data engineering for everyone.

Ruf, P., Madan, M., Reich, C., & Ould-Abdeslam, D. (2021). Demystifying mlops and presenting a recipe for the selection of open-source tools. *Applied Sciences*, *11*(19), 8861. doi: 10.3390/app11198861

Saltz, J. S., Yilmazel, S., & Yilmazel, O. (2016). Not all software engineers can become good data engineers. *2016 IEEE International Conference on Big Data (Big Data)*. doi: 10.1109/bigdata.2016.7840939

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... et al. (2015a). Hidden technical debt in machine learning systems. *Proceedings of the 28th International Conference on Neural Information Processing Systems*, *2*.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... Dennison, D. (2015b). Hidden technical debt in machine learning systems. In *Proceedings of the 28th international conference on neural information processing systems - volume 2* (p. 2503–2511). Cambridge, MA, USA: MIT Press.

Senapathi, M., Buchan, J., & Osman, H. (2018). Devops capabilities, practices, and challenges. *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*. doi: 10.1145/3210459.3210465

Shinde, P. P., & Shah, S. (2018). A review of machine learning and deep learning applications. *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*. doi: 10.1109/iccubea.2018.8697857

Smalling, E. (2021, Nov). *10 kubernetes security context settings you should understand.* Retrieved from `https://snyk.io/blog/10-kubernetes-security-context-settings-you-should-understand/`

Smedinga, R., & Biehl, M. (Eds.). (2020). *17th sc@rug 2020 proceedings 2019-2020*. Bibliotheek der R.U.

Soh, J., & Singh, P. (2020). Machine learning operations. In *Data science solutions on azure: Tools and techniques using databricks and mlops* (p. 259–279). Apress.

Stallman, R. (2009). *Floss and foss - gnu project - free software foundation.* Retrieved from

https://www.gnu.org/philosophy/floss-and-foss.html

Stallman, R. (2015). *Free software free society: Selected essays of richard m. stallman*. Free Software Foundation.

Tamburri, D. A. (2020). Sustainable mlops: Trends and challenges. *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. doi: 10.1109/synasc51798.2020.00015

Tang, Y., Khatchadourian, R., Bagherzadeh, M., Singh, R., Stewart, A., & Raja, A. (2021). An empirical study of refactorings and technical debt in machine learning systems. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. doi: 10.1109/icse43902.2021.00033

Vasconcelos, R. (2020, Jun). *Demystifying kubeflow pipelines: Data science workflows on kubernetes – part 1.* Ubuntu. Retrieved from https://ubuntu.com/blog/data-science-workflows-on-kubernetes-with-kubeflow-pipelines-part-1

Vasconcelos, R. (2021, Apr). *What is kfserving?* Ubuntu. Retrieved from https://ubuntu.com/blog/what-is-kfserving

Zhao, Y. (2020).

Ågerfalk, P. J., & Fitzgerald, B. (2008). Outsourcing to an unknown workforce: Exploring opensurcing as a global sourcing strategy. *MIS Quarterly*, *32*(2), 385. doi: 10.2307/25148845

Appendix A

1 Local Deployment Documentation

The local deployment of the single-user version of the designed MLOps lifecycle has constituted the basis for the fulfillment of this project. While it is not an enterprise-ready version, it provides sufficient data and an overview of how the final solution should look. Using the following listings, one can:

- Deploy the JupyterHub along with Kale and get a similar to DnA Platform's notebooks environment
- Deploy the Kubeflow Pipelines (single-user version) and get an overview of the interaction between the Jupyter Notebooks and KFP
- Deploy KServe and get an overview of the interaction between Jupyter Notebooks, KFP, and KServe

More information about this project can be found at its dedicated GitHub repository: `https://github.com/konsloiz/masters-thesis`

1.1 JupyterHub & Kale

Listing 1: Single User Jupyter Notebook & Kale Dockerfile

```
ARG IMAGE_TYPE="cpu"
FROM jupyter/pyspark-notebook:ubuntu-18.04

USER root
```

```
# Install basic dependencies
RUN apt-get update && \
   apt-get install -y --no-install-recommends \
      ca-certificates bash-completion tar less \
      python-pip python-setuptools build-essential python-dev \
      python3-pip python3-wheel && \
   rm -rf /var/lib/apt/lists/*

RUN apt-get install -y git

#RUN pip3 install jupyterlab-gitlab==2.0.0
# Install latest KFP SDK
RUN pip3 freeze
RUN pip3 install --upgrade pip && \
   # XXX: Install enum34==1.1.8 because other versions lead to errors
      during
   # KFP installation
   pip3 install --upgrade "enum34==1.1.8" && \
   pip3 install jupyterlab-gitlab==2.0.0 && \
   pip3 install --upgrade "jupyterlab>=2.0.0,<3.0.0"

#Install libraries for the demo
RUN pip3 install --upgrade pip && \
   pip3 install pillow==7.2.0 && \
   pip3 install tensorflow==2.3.0 && \
   pip3 install matplotlib==3.3.1 && \
   pip3 install torch &&\
   pip3 install torchvision

# Install Kale from KALE_BRANCH (defaults to "master")
ARG KALE_BRANCH="master"
WORKDIR /
RUN git config --global http.sslverify false
RUN git clone -b ${KALE_BRANCH} https://github.com/kubeflow-kale/kale
```

```
WORKDIR /kale/backend
RUN pip3 install --upgrade .

WORKDIR /kale/labextension
RUN npm config set strict-ssl false && \
   npm install --global yarn && \
   yarn config set "strict-ssl" false && \
   jlpm install && \
   jlpm run build && \
   jupyter labextension install .

RUN jupyter labextension install @jupyterlab/git
RUN pip3 install jupyterlab-git==0.24.0
RUN pip install nbgitpuller
RUN jupyter lab build

USER ${NB_UID}
WORKDIR "${HOME}"
```

Listing 2: JupyterHub Helm Chart Config

```
proxy:
  secretToken: #enter your token here
singleuser:
  image:
    name: # enter the image name here
    tag: latest
  defaultUrl: "/lab"
  extraEnv:
    KF_PIPELINES_ENDPOINT: http://ml-pipeline-ui.kubeflow:80
   #KUBECONFIG: /tmp/config
hub:
  extraConfig:
    ipaddress: |
     import os
     c.KubeSpawner.service_account = "hub"
```

Listing 3: JupyterHub & Kale Deployment

```
helm install jupyter jupyterhub/jupyterhub --version=v0.11.0 -f config.
    yaml -n kubeflow --timeout 180s

kubectl port-forward -n kubeflow svc/proxy-public 8888:80
```

1.2 Kubeflow Pipelines - Single User Version

Listing 4: Kubeflow Pipelines Deployment

```
kubectl apply -k github.com/kubeflow/pipelines/manifests/kustomize/
    cluster-scoped-resources?ref=1.7.0

kubectl wait --for condition=established --timeout=60s crd/applications
    .app.k8s.io

kubectl apply -k github.com/kubeflow/pipelines/manifests/kustomize/env/
    dev?ref=1.7.0

kubectl port-forward -n kubeflow svc/ml-pipeline-ui 3000:80
```

1.3 KServe

Listing 5: KServe Deployment

```
curl -s "https://raw.githubusercontent.com/kserve/kserve/release-0.7/
    hack/quick_install.sh" | bash

kubectl delete -f https://github.com/kserve/kserve/releases/download/v0
    .7.0/kserve.yaml

kubectl apply -f https://github.com/kubeflow/kfserving/releases/
    download/v0.6.0/kfserving.yaml

# set the following ENV vars in the app's Deployment
kubectl edit -n kfserving-system deployments.apps kfserving-models-web-
```

```
    app
# APP_PREFIX: /
# APP_DISABLE_AUTH: "True"
# APP_SECURE_COOKIES: "False"

# expose the app under localhost:5000
kubectl port-forward -n kfserving-system svc/kfserving-models-web-app
    5000:80

# authorize network access to deployment
kubectl port-forward svc/istio-ingressgateway -n istio-system 8080:80
```