FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Cloud deployment using MLOps tools
Student:	Aleš Sršeň
Supervisor:	Ing. Tomáš Vondra, Ph.D.
Study program:	Informatics
Branch / specialization:	Software Engineering 2021
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2024/2025

Instructions

The development lifecycle of machine learning models differs significantly from the traditional software development lifecycle. This thesis aims to migrate an ML application provided by Profinit to a cloud environment and design and implement an MLOps pipeline that will cover the MLOps lifecycle (development, model training, data management, model tracking, and application deployment). As a result, a comparison between the original and the new approach will be made.

1. Analyze the current state of the application.
2. Understand and describe MLOps principles.
3. Research and choose appropriate tools to use in the pipeline.
4. Design the pipeline and migrate the application to the cloud.
5. Evaluate and compare the newly implemented pipeline with the original state of the application.

Electronically approved by Ing. Michal Valenta, Ph.D. on 26 January 2024 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Cloud deployment using MLOps tools

Aleš Sršeň

Department of Software Engineering
Supervisor: Ing. Tomáš Vondra, Ph.D.

May 16, 2024

Acknowledgements

Firstly, I would like to express my deepest gratitude to my supervisor, Ing. Tomáš Vondra, Ph.D., for all the guidance, consultation, and valuable feedback. Secondly, this endeavor would not have been possible without the company Profinit EU and its people. I want to thank them for providing the thesis topic and cloud resources for the implementation. Finally, I want to thank my family for their support during the work on this thesis.

Declaration

Citation of this thesis

Sršeň, Aleš. *Cloud deployment using MLOps tools*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Abstract

With Machine Learning (ML) gaining broader adoption and popularity, a new paradigm focused on applying DevOps methodology to ML systems, known as Machine Learning Operations (MLOps), has emerged. This thesis investigates the application of the MLOps paradigm to the development life cycle of an application utilizing ML provided by Profinit EU. In this thesis, MLOps is introduced, its principles are described, and an overview of select MLOps tools is provided. Based on this information, a pipeline to apply MLOps principles is designed. The provided application is migrated to a cloud environment, where the designed pipeline is implemented and used with the application.

Keywords MLOps, machine learning, cloud computing, DevOps, pipeline

Abstrakt

S rastúcou popularitou strojového učenia (ML) sa objavila nová paradigma zameraná na aplikáciu metodiky DevOps na systémy ML, známa ako Machine Learning Operations (MLOps). Táto práca skúma aplikáciu paradigmy MLOps na životný cyklus vývoja aplikácie využívajúcej ML, poskytnutej spoločnosťou Profinit EU. V práci je predstavená MLOps paradigma, opísané jej princípy a uvedený prehľad vybraných nástrojov MLOps. Na základe týchto informácii je navrhnutá pipeline na aplikovanie princípov MLOps. Poskytnutá aplikácia je migrovaná do cloudového prostredia, kde je navrhnutá pipeline implementovaná a použitá s aplikáciou.

Klíčová slova MLOps, strojové učenie, cloud computing, DevOps, pipeline

Contents

List of Figures

Introduction

Machine learning roots date back to the 1950s when Arthur Samuel invented a program calculating each side's winning chance in checkers. Since then, this field has come a long way and has been unwinding faster and faster in the last few years.

The advancements in this field have enabled us to use computers to accomplish tasks that were once thought impossible for them to achieve. From the perspective of many researchers, the exploration of the capabilities of machine learning is not showing signs of slowing down or stopping. To quote Dave Waters, *"A baby learns to crawl, walk and then run. We are in the crawling stage when it comes to applying machine learning."* [1].

In the same way, the baby who is learning to crawl needs to be taken care of and looked after; machine learning needs to be looked after. The company Profinit EU is fully aware of this fact and wants to get its machine learning projects to crawl successfully. That is why they have offered to support research and experiments in a new field called machine learning operations, or MLOps for short. MLOps has emerged based on machine learning needs and the needs of the people involved.

Machine learning operations are still in their infancy, so together with Profinit EU, we see the need to experiment and explore this field to gain knowledge and share it with others. Hopefully, this thesis will unveil the intricacies of teaching the baby to crawl.

To apply our newly gained knowledge, an application utilizing machine learning is provided by Profinit EU. We hope to be able to evaluate our strategy and choices using this application and try to apply the MLOps principles when deploying the application to the cloud services.

At the beginning of this thesis, the goal is to see and understand the ideas of more experienced individuals. As mentioned before, since this field is relatively new, it is essential to consider many opinions to get the whole picture. Later, some of the tools in the vast landscape of MLOps tools will be described, and some will be chosen based on our needs. Sometimes, the choices may be excessive for our current use case, but they can be seen as necessary to gain universally applicable knowledge. After choosing the tools for the job, the plan is to apply and integrate those tools to get the best experience possible. Finally, we will look back and evaluate our solution to problems posed by machine learning and reflect on the freshly gained insight into the new field of MLOps.

To summarize, this thesis strives to advance the field of machine learning by addressing the problems with the development and operations of machine learning models and improving the state of the application provided by Profinit EU along the way.

Thesis's Objective

The primary goal of this thesis is to get a good grasp of the MLOps paradigm and its principles and later utilize them in designing a pipeline using tools intended to help successfully implement this paradigm. The designed pipeline will be implemented and used with an application provided by the company Profinit EU.

At the beginning of this thesis, the reader will be introduced to the MLOps paradigm and its principles. Since MLOps is the set of practices applied to machine learning projects, we will introduce machine learning and parts of its life cycle. Next, we will look at the tools available in the MLOps ecosystem and choose the ones we see fit based on our needs and criteria. Following that, there will be an overview of the application provided by Profinit EU.

Towards the end of this thesis, a pipeline that covers most of the steps involved in successfully applying MLOps will be designed. The designed pipeline will be implemented in a cloud environment and used with the provided application, which will also be migrated to a cloud environment. After achieving the goals we set out to do, we will reflect on our choices and offer our opinion on how the solution could be enhanced or adjusted.

Chapter 1

Machine Learning Operations

At the beginning of this chapter, we will define machine learning and look at the main differences between machine learning and traditional software development. Based on the literature, we will highlight the challenges of machine learning. Next, we will look at what steps a machine learning life cycle can be comprised of and learn about a formalized process model to improve the success of machine learning projects. Lastly, we will define the main focus of the thesis, machine learning operations, and its principles.

1.1 Machine Learning

Machine Learning (ML) is a field of computer science focused on the development of algorithms that allow computers to learn and improve from data [2]. With the trend of collecting data growing exponentially [3], ML has proven to be a powerful tool for a wide range of problems. It is often used to solve tasks that can be too complex to be solved traditionally. The main thing that differentiates ML from traditional software is that instead of telling your computer how to transform the data, you can provide the outputs and let your computer figure out the pattern using an ML model [4] as shown in the diagram 1.1.

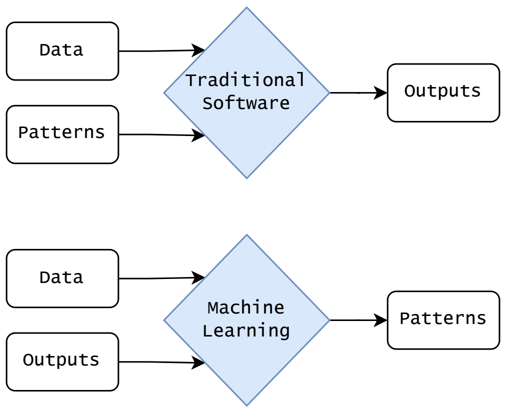


Figure 1.1: The computer learns the patterns instead of requiring hand-specified patterns to calculate the outputs [4].

The difference between traditional software and ML allows the former to use strong abstraction boundaries, encapsulation, and modular design, which help create maintainable code [5]. However, the use of ML is necessary when the desired behavior cannot be expressed in software logic without relying on external data [6].

For us, this stresses that ML projects are composed of not just the model. Many exceptional researchers argue that the data is the most critical part of ML projects [7].

After considering the data and model, there is also the need to integrate the ML model into existing software using code. This outlines the three levels of change in ML projects – data, model, and code – where the trigger for a build might come from [8]. One of the first papers focused on highlighting the topic of technical debt in ML gave a name to this principle: Changing Anything Changes Everything (CACE). [6]

1.1.1 Challenges with ML

The book *Introducing MLOps* summarizes the three key reasons why managing ML projects life cycle at scale is challenging [9]:

- **Dependencies.** As can be seen, there are many dependencies when developing an ML project. Data is not the only thing constantly changing; often, business needs shift with time. Results must be monitored and analyzed to ensure the expectations meet the original goal.
- **Language barrier.** ML project life cycle involves many people from different areas. These people often do not share the fundamental skills that enable efficient communication.
- **Staff skills.** Data scientists are the main ingredient in ML project development. The thing is that they are specialized in model building and assessment rather than software development. Though this may change over time, juggling many roles is often problematic.

Together, these areas pose a reason why most of the ML projects started in companies never get deployed in the production [10].

1.2 Machine Learning Life Cycle

To get a feel for the steps that need to be taken to develop ML projects, we will look at a high-level overview of ML life cycle and then introduce a formalized process model. The actual steps taken in practice can vary significantly based on the exact project requirements.

1.2.1 High-level Overview of ML Life Cycle

ML projects aim to build a statistical model by applying machine learning algorithms to collected data and then provide the created model or use its insights. This goal outlines a ML life cycle that can be divided into three high-level phases: data engineering, model engineering, and model deployment. Each of these three phases includes multiple steps that lead to the successful creation of ML projects. [11]

1.2.1.1 Data Engineering

For any data science project, acquiring and preparing the data for analysis is necessary. The data can come from many sources in a variety of different formats. In this phase, insights have to be gained from this data, and the goal is to create training, validation, and testing datasets for the ML algorithms. This phase might be comprised of the following steps [12, 13]:

- **Data collection.** First, the data needs to be collected using various frameworks and formats. The data can also be generated or artificially enriched.
- **Data exploration.** The data needs to be profiled, and information about the content and structure of the data needs to be obtained. Data scientists use many metrics and visualizations of the data to gain knowledge and intuition about the characteristics of the data. This intuition is then used to choose the right methods in later steps.
- **Data wrangling.** This step is comprehensive and may involve various activities. These include restructuring the obtained data, cleaning the dataset, engineering new features, and enriching the data with additional context, among others.
- **Data validation.** To ensure there are no errors in the data, it should be validated before continuing to the next phases. Skipping this step can lead to failure to achieve the goal in the next phases.
- **Data labeling.** For supervised learning applications, the data points need to be labeled to derive insights from the data later. This process can be done manually or automated with other tools using machine learning. Wrong labels can lead to incorrectly derived insights from the data later.
- **Data splitting.** To be able to train the machine learning models and later validate their performance, the obtained data should be split into training, validation, and test datasets. The training dataset should contain enough information for the model to derive insights and represent the dataset as a whole. A validation dataset is used for hyperparameter optimization, influencing the model directly. The test dataset is separate from the whole process to get an independent overview of the model's performance [14].

1.2.1.2 Model Engineering

The model engineering phase can start after the data is prepared in the data engineering phase. Model engineering is the core phase of the ML life cycle where the engineers write and execute ML algorithms to obtain an ML model. The algorithms are provided with the data obtained and processed in the previous phase. The goal is to get a model that is the best possible for our use case. We can achieve that by taking the following steps [12, 15]:

- **Model selection and training.** In this step, the engineer needs to design or select a model architecture that provides the best results. Multiple models are often chosen at first, and some may already be pre-trained.

Then, they are trained and validated to find the best candidate. Based on the validation results, the parameters of the models are optimized in a process called hyperparameter optimization. After choosing the best parameters, the model is then trained until the performance goal or other conditions are met [14]. This final training is often the most resource-intensive part of the ML life cycle.

- **Model testing.** After the final training of the model with the optimized parameters, the model needs to be evaluated and tested to see the generalization error on unseen data that have not had any impact on the model training. If the error rate is too high or the business objectives are not met, it is necessary to re-initiate the model selection and training step again.
- **Model packaging.** When the tests are successful, the model needs to be serialized and packaged with the consideration of how it is going to be deployed. There are multiple possible package formats, each with its pros and cons [16].

1.2.1.3 Model Deployment

To successfully deliver an ML project, the model must be somehow provided to the application that utilizes it. This can be done in various ways. After providing the model, this phase is still not finished since it is necessary to monitor the model to observe its performance in production. The model deployment phase may include these steps:

- **Model serving.** Depending on the context of the application, there are multiple approaches to serving the final model. For example, the model can be available as a service, dependency of the final application, the predictions can be precomputed and stored in a database, and many others. Based on the needs, the model can be served from different environments such as ML platform, Kubernetes cluster, serverless functions, and others. [12]
- **Model monitoring.** After serving the model, we need to monitor it to assess its performance in production. There are multiple performance metrics that can be monitored. For example, monitoring should detect data and concept drifts since they often have a big impact on the model's performance. Data drift covers the changes to the data distribution of inputs compared to training data input distribution. Concept drift is when the properties of the target variable change. According to the monitoring results, the engineers should investigate and take the necessary steps, or an automated action can be performed. [17]

1.2.2 CRISP-ML(Q) – ML Process Model

The machine learning life cycle is often a complex process, which is why it can benefit from standardization. With the rising interest in this field, efforts were made to create a standard process model.

In 2019, a conference paper by *Amershi et al.* [18] described a machine learning life cycle of projects at Microsoft. This was later revised and built upon by a 2021 paper by *Studer et al.* [19], which proposed a process model for the development of machine learning expanding on CRISP-DM[1] model.

This process model is called CRoss-Industry Standard Process model for the development of Machine Learning applications with Quality assurance methodology (CRISP-ML(Q)). It proposes six phases displayed in diagram 1.2, from defining the scope to maintaining the deployed ML application.

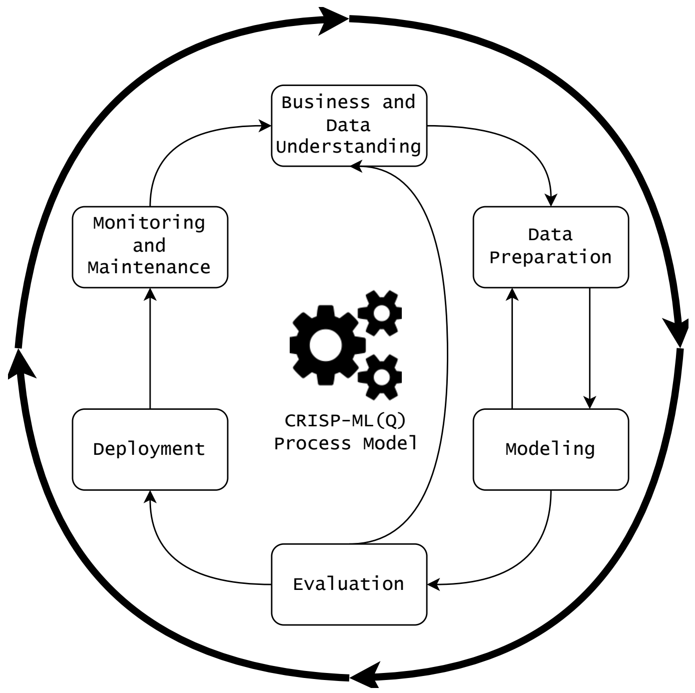

Business and Data Understanding
Data Preparation
Modeling
Evaluation
Deployment
Monitoring and Maintenance
CRISP-ML(Q) Process Model

Figure 1.2: Illustration of the six phases of CRISP-ML(Q) process model [19].

For each phase of the model, the quality assurance approach requires the definition of requirements and constraints. After that, the specific tasks can be instantiated. In each task, specific risks that might negatively impact the efficiency and success of ML project need to be identified. Quality assurance methods are used to mitigate these risks. [19]

Each of the six phases is described below in some detail to show what the phase's concern is.

1.2.2.1 Business and Data Understanding

The initial phase concerns defining the business objectives, translating them to ML objectives, collecting and verifying the data quality, and assessing the project feasibility. In this step, measurable success criteria should be defined. Based on these criteria, decisions should be made in the process's later steps.

[1] Proposed standard process model for data mining. See paper [20].

1.2.2.2 Data Preparation

Using the insights from the preceding data understanding phase, this phase produces a data set for the following modeling phase. During this phase, the data is selected and cleaned. New features are derived from the existing ones and standardized.

1.2.2.3 Modeling

This is an ML-specific part of the process. This phase aims to specify one or several models to be deployed in the production. Generally, it includes model selection, model specialization, and model training tasks. Various metadata should be collected to ensure the method and results are reproducible. If the modeling phase reveals erroneous data, backtracking to the data preparation phase is necessary.

1.2.2.4 Evaluation

The model engineering phase is followed by a model evaluation phase, often known as offline testing. The performance of the model needs to be examined on a test set. Checking the model's robustness using noisy or wrong input data is also preferred.

Documenting all outcomes of this phase is recommended, as is the case with the preceding phases. The model deployment decision should be made by domain experts using the success criteria defined in previous phases. If the developers cannot meet the success criteria, a decision to backtrack to earlier phases or stop the project may be in place.

1.2.2.5 Deployment

The deployment phase focuses on the process of ML model integration into the existing software system. Deployment approaches differ depending on the use case and the fashion of prediction. Most often, predictions are done in batches or online.

Tasks in this phase include evaluating hardware requirements for inference, model evaluation in the production environment, testing, planning for outages, and setting up the deployment strategy for new model versions.

1.2.2.6 Monitoring and Maintenance

Without maintaining and monitoring the model, there are risks of performance degradation. This can lead to false predictions and can cause errors in the subsequent systems depending on those predictions. The main risk is the "model staleness" effect, which occurs when the model's performance drops as it starts operating on unseen data.

The best practice to avoid performance drops is to monitor and maintain the model by evaluating the results and deciding whether re-training is necessary. [19]

1.3 Machine Learning Operations

To address at least some of the challenges of developing ML projects, a new paradigm called Machine Learning Operations (MLOps) has emerged, combining the practices from different fields: Machine Learning, DevOps, and Data Engineering [21].

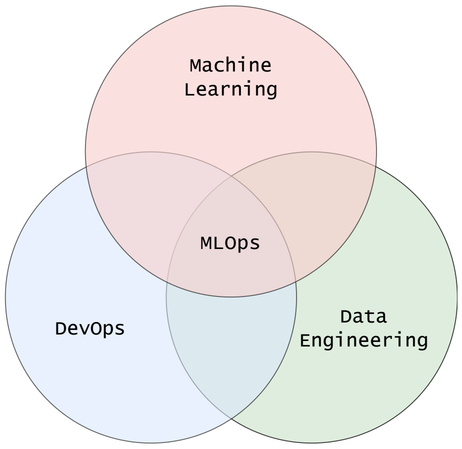


Figure 1.3: MLOps combines practices from different fields [21].

Much like other emerging technologies, there is no agreed-upon definition for the term MLOps. The literature focused on conceptualizing this term often highlights that MLOps is a paradigm aimed at getting the machine learning projects to production by bridging the gap between Development (Dev) and Operations (Ops) [22, 21]. Some go as far as to summarize it as applying DevOps methodology to ML systems [23]. Similarly to DevOps, it is characterized as streamlining and standardization of ML life cycle management [9].

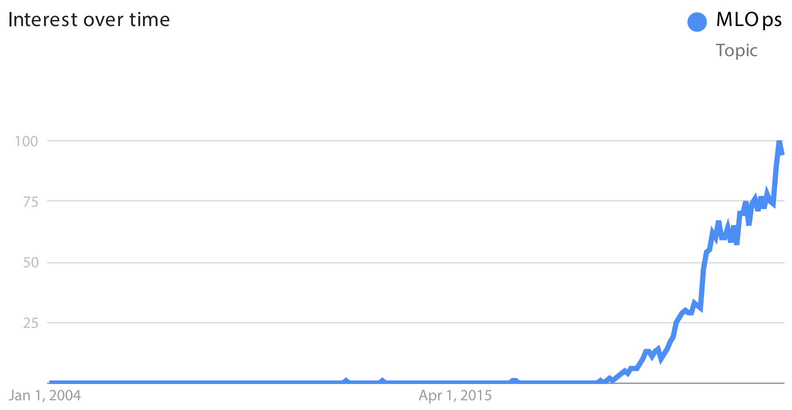


Figure 1.4: Interest over time in MLOps based on Google Trends data [24].

This paradigm is quickly becoming a critical component in successful ML project development. It is also reflected by the growing popularity of this paradigm captured by Google Trends in figure 1.4.

As mentioned before in the section 1.1 introducing ML, MLOps is trying to tackle the complexity arising from the fact that ML projects are made up of code, data, and models. This makes it the critical difference between MLOps and DevOps, which is why the latter is not immediately transferable to the development of ML projects.

Much like other paradigms, there is no one-fits-all way to implement MLOps paradigm. What MLOps does is that it provides the practitioners with insights and best practices to increase efficiency and help achieve business goals faster. Unfortunately, due to the age and the broad scope of the life cycle of ML projects, there are often discrepancies in the literature concerning the best practices and application of this paradigm. [25]

1.3.1 DevOps

To get a better picture, let us look at DevOps software development methodology. DevOps combines Development (Dev) and Operations (Ops) to increase the efficiency, speed, and security of software development and delivery of the product compared to traditional processes. This methodology provides a competitive advantage thanks to being more nimble. [26]

The DevOps methodology aims to shorten the development life cycle and provide continuous delivery with high quality. It comprises four key principles that guide application development and deployment effectiveness. The key principles dictate the aspects of modern software development:

- **Automation of the software development life cycle.** It is necessary to automate testing, builds, releases, and other manual tasks that slow down the delivery process or can introduce human error.
- **Collaboration and communication.** Effective communication and collaboration are essential in DevOps.
- **Continuous improvement and minimization of waste.** To enhance the efficiency, improvements should be made constantly. Teams should regularly look for ways to avoid wasting time and enhance the process.
- **Hyperfocus on user needs with short feedback loops.** Using automation, improved communication and collaboration, and continuous improvement, DevOps teams should focus on what users want and how to provide it. [26]

1.3.2 Data Engineering

Data engineering is at the root of most of the ML projects. This process focuses on designing and building systems to collect and analyze raw data from various sources and multiple formats. Data is essential for businesses since it enables them to derive insights to achieve their business goals. It is concerned with many different tasks, such as acquisition, cleaning, conversion, storage, and the architecture of data pipelines. [27]

1.4 Principles of MLOps

As mentioned before, due to the MLOps paradigm being relatively new and broad in scope, it is not well formalized and agreed upon what are the main principles of MLOps. To provide the reader with the overview of MLOps principles, we have chosen to use the description of the principles from the paper by *Kreuzberger et al.* [22] where a mixed-method research was conducted, including a literature review, a tool review and expert interviews. The description of some of these principles is expanded by information from other sources.

In the context of MLOps, a principle is viewed as a guide on how to apply said paradigm. The paper mentioned above outlines nine principles described below [22]:

- **CI/CD Automation.** Continuous Integration (CI) and Continuous Delivery (CD) steps such as build, test, delivery, and deployment should be automated. CI/CD puts principles of DevOps into practice, with the CI/CD pipeline being considered the backbone of DevOps. This principle provides the developers with quick feedback regarding the results of certain life cycle steps.
- **Workflow orchestration.** ML workflow pipeline tasks can be organized to Directed Acyclic Graphs (DAGs). Considering the relationships and dependencies, DAGs define the order of task execution. Workflow orchestration coordinates the ML workflow according to the DAGs.
- **Reproducibility.** In the machine learning project, it should be possible to reproduce an ML experiment and obtain the same results. We want to avoid non-determinism and store the values of variable inputs.
- **Versioning.** Versioning is essential to ensure reproducibility and traceability. These are often necessary for compliance and auditing reasons. Everything from data to model and code should be versioned.
- **Collaboration.** It should be possible for the developers to work collaboratively on the data, model, and code for the ML project. Collaboration aims to reduce domain silos between different roles by emphasizing communication and collaborative work.
- **Continuous training and evaluation.** The need for continuous training and evaluation stems from changes in the data. It is enabled by a monitoring component, a feedback loop, and an automated ML workflow pipeline. It is necessary to include an evaluation run after training the model to assess the change in model quality. [22] To avoid high costs associated with the continuous retraining of the model, it should be carefully considered how often the retraining should happen or use online machine learning, where the training steps are iterative compared to training on full data set [28].
- **ML metadata tracking/logging.** In order to help with traceability, the information about each execution of the ML pipeline should be recorded. This also helps with debugging errors and anomalies. For each run, it is helpful to store the following metadata: the code of the pipeline, date and

time of the execution, duration of the pipeline run, model parameters, and model performance metrics; it would also be beneficial to store the data and artifacts from the pipeline, or at least pointers to them. [29]

- **Continuous monitoring.** Since ML models' performance can degrade for various reasons, we need to monitor it. Monitoring does not need to be limited only to the model since the errors can appear in other parts of the ML project. The monitoring statistics can act as a trigger for a new pipeline run or to begin experimenting again. Retraining should lead to model recovery. [17]

- **Feedback loops.** To ensure the quality of models in the model engineering phase, insights from the quality assessment steps might require changes in the data engineering phase. This necessitates including feedback loops in the ML development process. Another feedback loop is needed from the production environment, facilitated by the monitoring component, to developers or to trigger the retraining as mentioned in the previous principles. [9]

These principles form an overview of what an effective ML project should include. The more principles are implemented in the ML development process, the more streamlined it should become.

In the next chapter, we will look at some of the tools that help implement the principles.

Chapter 2

MLOps Tools and Frameworks

After gaining knowledge of what MLOps is and the principles of MLOps that should be applied in practice, we will get an overview of tools available in the MLOps landscape and their usage.

Many of the tools in the MLOps landscape do not focus solely on one goal, which is why they often have overlapping features. It is challenging to categorize them into distinct groups. We have split the tools based on their primary goal – the feature they focus on the most.

In the past few years, many open-source and closed-source tools were developed and made available to the public. Open-source solutions have gained much traction thanks to their flexibility, community support, and adaptability to various workflows. On the other hand, closed-source platforms often provide enterprise-grade features, enhanced security, and dedicated user support.

2.1 Criteria for Selecting MLOps Tools

The selection of tools discussed in this chapter is guided by a set of criteria established in collaboration with Profinit EU. A primary objective of this collaboration is to avoid the issue of vendor lock-in – a scenario where a customer becomes so reliant on a particular vendor's products and services that switching to another vendor becomes excessively expensive, both financially and in terms of time and resources [30]. To avoid this risk, we narrowed our search to open-source tools that can be self-hosted without any licensing costs. This approach ensures that Profinit EU can maintain flexibility and independence from any single vendor's ecosystem.

The landscape of MLOps tools has become exceedingly vast, so we will also consider the tool's popularity and maturity. Choosing a relatively new tool may prove problematic since more breaking changes may be made than with mature and more popular tools.

2.2 End-To-End MLOps Platforms

To avoid dealing with the issues related to integrating multiple products in the MLOps pipeline, it might be worth looking at end-to-end MLOps platform offerings. These provide a unified ecosystem, focusing on the entire ML workflow, from the early phase of data engineering through model engineering, and provide tools to enable deployment with monitoring [31].

End-to-end MLOps platforms are offered mostly as closed-source enterprise platforms such as Amazon SageMaker, Microsoft Azure Machine Learning, Google Cloud Vertex AI, and many others. We chose to avoid these platforms for the reasons described in the section 2.1.

2.2.1 Core Features

Features of the end-to-end MLOps platforms can vary significantly based on the chosen platform. The features pointed out below are some that should be included in the platform. In the next few sections, we will look at some components that should be parts of the end-to-end MLOps platforms, and their more specific features should be included in these core features as well.

- **Version control.** Building on the described MLOps principles, the platform should enable version control for various ML artifacts. This helps to ensure reproducibility and collaboration, simplifying the process of sharing artifacts.

- **Data management and preprocessing.** The platform should provide capabilities for the data engineering phase of the ML development process. The steps that should be supported by the end-to-end MLOps platform include data ingestion, storage, preprocessing, labeling, versioning, and others described in the subsection 1.2.1.1.

- **Workflow orchestration.** Tools for automated workflow orchestration should be included in the platform to automate the processes tied to data engineering and model training. The workflow orchestration component should enable dependency management, task scheduling, error handling, and overall simplification of the management of ML workflows.

- **Experimentation and model development.** Developers should have available tools to support the model engineering phase described in the subsection 1.2.1.2. These tools should enable the developer to run experiments, explore different algorithms and architectures, compare their performance, and find the optimal parameters of the model. This should be enabled by using tools for hyperparameter tuning, automated model selection, and visualization of model metrics.

- **Model deployment and serving.** Models built in the model engineering phase should be easily deployed using features provided by the platform, such as containerization, API management, and infrastructure scaling based on demand.

- **Model monitoring and performance tracking.** Getting information about the model's declining performance in production is essential to ensure reliability. That is why the platform should provide capabilities to monitor and track the performance of the deployed ML model. This includes tools for logging, monitoring model metrics, detecting anomalies, and alerting.

- **Integration with ML tools and libraries.** It is important for the platform to be able to integrate with existing and new ML tools and libraries. This provides flexibility and extensibility by filling in the gaps in the development process. Developers can leverage their preferred tools and many resources to increase productivity and use new technologies. [31, 32]

2.2.2 Available Platforms

Most end-to-end MLOps platforms are closed-source or open-source only specific components of their implementation. For this reason, only one platform is described below. That platform is open-source and has all its main components available.

2.2.2.1 Kubeflow

Figure 2.1: Kubeflow logo [33].

Website	`www.kubeflow.org`
Creator	Google
Maintainer	Kubeflow Contributors
Release Year	2018
License	Apache License, Version 2.0

Table 2.1: The basic information on Kubeflow [33].

Kubeflow is an open-source project developed to make deployments of machine learning workflows on Kubernetes simple, portable, and scalable. It is closely tied to the Kubernetes platform, meaning Kubeflow can be deployed only on Kubernetes.

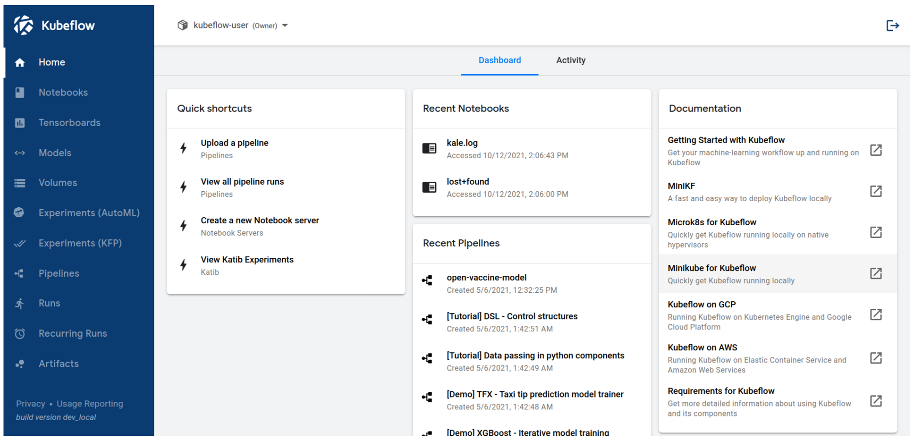
It implements a variety of ML tools in the form of components -— from data preprocessing to model training and serving —- and provides a unified interface to manage them. This allows data scientists and ML engineers to focus more on their models and less on infrastructure management.

The main components often distributed with the installations of Kubeflow are:

- Kubeflow Notebooks – Web-Based IDEs,
- Kubeflow Pipelines – Workflows/Schedules,
- Central Dashboard – Web Interfaces,

- Training Operator – Model Training,
- Katib – Model Tuning,
- KServe – Model Serving.

These components are not everything Kubeflow has to offer. Using external add-ons created by us or the community is also possible. This way, it is possible to add, for example, a feature store, different tools for serving models, visualisations…



Quick shortcuts

Upload a pipeline
Pipelines

View all pipeline runs
Pipelines

Create a new Notebook server
Notebook Servers

View Katib Experiments
Katib

Recent Notebooks

kale.log
Accessed 10/12/2021, 2:06:43 PM

lost+found
Accessed 10/12/2021, 2:06:00 PM

Recent Pipelines

open-vaccine-model
Created 5/6/2021, 12:32:25 PM

[Tutorial] DSL - Control structures
Created 5/6/2021, 1:42:51 AM

[Tutorial] Data passing in python components
Created 5/6/2021, 1:42:49 AM

[Demo] TFX - Taxi tip prediction model trainer
Created 5/6/2021, 1:42:48 AM

[Demo] XGBoost - Iterative model training

Documentation

Getting Started with Kubeflow
Get your machine-learning workflow up and running on Kubeflow

MiniKF
A fast and easy way to deploy Kubeflow locally

Microk8s for Kubeflow
Quickly get Kubeflow running locally on native hypervisors

Minikube for Kubeflow
Quickly get Kubeflow running locally

Kubeflow on GCP
Running Kubeflow on Kubernetes Engine and Google Cloud Platform

Kubeflow on AWS
Running Kubeflow on Elastic Container Service and Amazon Web Services

Requirements for Kubeflow
Get more detailed information about using Kubeflow and its components

Figure 2.2: Screenshot of the Kubeflow Central Dashboard UI [33].

As mentioned in the first few lines, Kubeflow is built on top of the Kubernetes platform. This gives Kubeflow all of the advantages that come with Kubernetes, from the ability to deploy on any infrastructure (where Kubernetes can run) to managing loosely coupled microservices and on-demand scaling. [33]

What is unfortunate is that Kubeflow is pretty complex, and a lot of its documentation is outdated. The loose coupling of its components comes with the benefit of choosing which components to use. However, different components may rely on different versions of the same dependencies, causing problems.

Advantages

+ Simple scalability using Kubernetes.
+ Portability across environments.
+ Large ecosystem.
+ Comprehensive ML toolkit with various tools for ML projects.

Disadvantages

- Setup and configuration are complex.
- Running Kubeflow at scale is resource-intensive.
- High maintenance overhead.
- Outdated parts of documentation.

2.3 Workflow Orchestration and Pipelining

Workflow orchestration and pipelining tools are essential for streamlining and automating complex ML workflows [31].

An ML workflow refers to the sequence of steps needed to be taken when building an ML project. We have outlined the possible steps in ML workflow in subsection 1.2.1. For example, the workflow can include data collection, data preprocessing, model selection, and model training. A pipeline is the implementation of the workflow.

Workflow orchestration and pipelining tools help automate and manage workflows and pipeline infrastructure. This is often facilitated by task orchestration using Directed Acyclic Graphs (DAGs). [22]

2.3.1 Core Features

The workflow orchestration and pipelining tools should provide the following features [31, 34]:

- **Dependency management.** Arguably, the most important feature of workflow orchestration tools is the ability to define dependencies between workflow tasks and automatically execute them in the correct order. It is necessary that all of the dependencies of a task are satisfied before executing it. Most of the tools define the workflows as DAG of tasks.

- **Workflow monitoring and visualization.** The workflow orchestration tool should provide workflow monitoring and visualization to get an idea of what is happening when running workflows. The monitored information can include evidence of the workflow runs, the progress of the running workflows, inputs and outputs of the workflows, and other information. Visualization of workflow task dependencies is also a helpful feature.

- **Workflow scheduling and triggering.** To be able to run workflows effectively and without manual intervention, the workflow orchestration tool should allow workflow scheduling and triggering using various sources. Scheduling (running the workflow on a set schedule) is most often provided by cron expressions. Webhooks are used to provide the ability to trigger workflows from various sources.

- **Reproducibility and versioning.** Two of the principles of MLOps outlined were reproducibility and versioning. Workflow orchestration tools should support reproducibility by providing a way to capture the workflow configuration with code, datasets, and dependencies included. The workflows should be ideally versioned so that each workflow run has the necessary artifacts to reproduce it.

- **Error handling and retry mechanisms.** Workflow orchestration tools should handle errors gracefully, ensuring the reliability and robustness of ML workflows. The tools should provide retry mechanisms to handle failures and retry failed tasks. Failures should provide information to the developers so they are able to debug failed workflows.

- **Distributed computing and scalability.** With the ML being applied to larger and larger tasks, the workflow orchestration tool should provide the necessary scaling to handle resource-intensive workflows. Scaling is most often done by distributing the computing tasks and requesting additional resources necessary for the workflows.

2.3.2 Available Tools

Nowadays, multiple open-source tools are available in the workflow orchestration space. Workflow orchestration can be applied to other fields, not just ML, so there are tools that are not just ML-specific. We will limit our extensive overviews to tools that consider the needs of ML and provide information about a select number of tools.

2.3.2.1 Flyte

Figure 2.3: Flyte logo [35].

Website	`www.flyte.org`
Creator	Lyft
Maintainer	Flyte
Release Year	2019
License	Apache License, Version 2.0

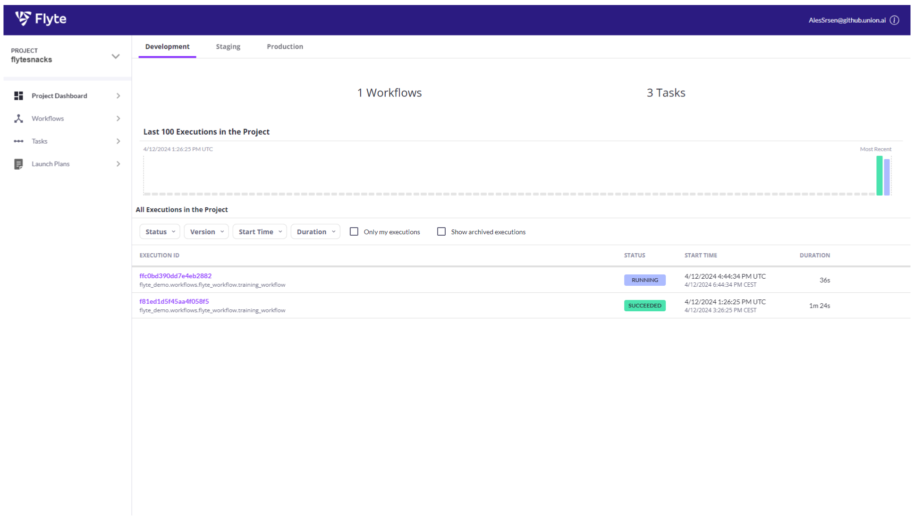
Table 2.2: The basic information on Flyte [35].

Flyte is a powerful open-source Kubernetes-native workflow orchestrator designed to streamline the development and execution of data-intensive workflows, especially in the fields of data science and machine learning. It provides libraries in multiple languages, including Python, Java, Scala, and JavaScript.

It allows for very complex workflows, which in turn means there is a steep learning curve associated with this tool. In Python, workflows are defined using decorators provided by Flyte's library. Flyte also provides integrations with various popular MLOps tools.

Flyte allows data flow tracking to provide data lineage, enables dynamic workflow definition, and provides data visualization. These are all very valuable features for the developers. Workflows can be scheduled, run from the UI, or triggered remotely.

Development with Flyte can start in a local setting and seamlessly transition to production environments on major cloud platforms like AWS, GCP, and Azure. The orchestrator supports cloud-native deployments, which allows leveraging cloud-specific features such as spot instances and GPU acceleration for cost efficiency and performance enhancement. [35]



1 Workflows

3 Tasks

Last 100 Executions in the Project

4/12/2024 1:26:25 PM UTC

All Executions in the Project

EXECUTION ID	STATUS	START TIME	DURATION
ffc0bd390dd7e4eb2882 flyte_demo.workflows.flyte_workflow.training_workflow	RUNNING	4/12/2024 4:44:34 PM UTC 4/12/2024 6:44:34 PM CEST	36s
f81ed1d5f45aa4f058f5 flyte_demo.workflows.flyte_workflow.training_workflow	SUCCEEDED	4/12/2024 1:26:25 PM UTC 4/12/2024 3:26:25 PM CEST	1m 24s

Figure 2.4: Screenshot of the Flyte UI.

Advantages

+ High workflow scalability.
+ Integrations with other MLOps tools.
+ Powerful workflow control features.

Disadvantages

- Steep learning curve.
- Deployment is resource intensive.

2.3.2.2 Metaflow

Figure 2.5: Metaflow logo [36].

Website	`www.metaflow.org`
Creator	Netflix
Maintainer	Outerbounds
Release Year	2019
License	Apache License, Version 2.0

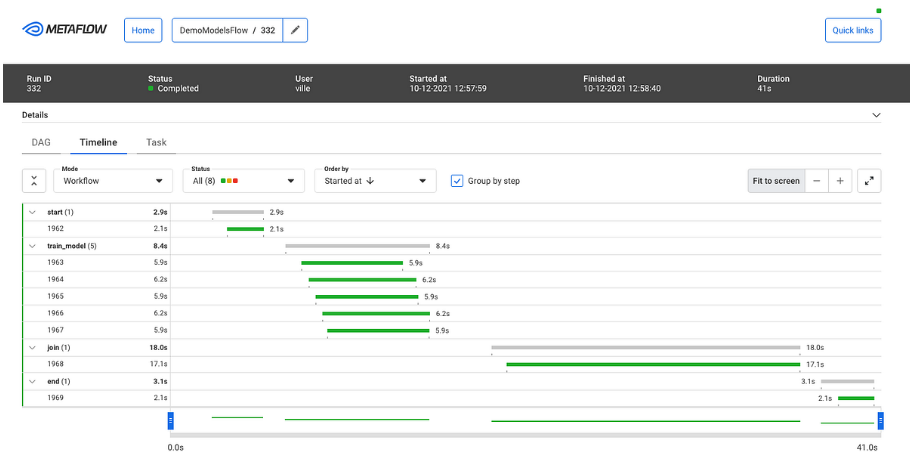
Table 2.3: The basic information on Metaflow [36].

Metaflow is an open-source workflow orchestration framework that makes the process of developing and deploying data-intensive, particularly machine learning, workflows straightforward. The interaction with Metaflow is through a library that provides a unified API that offers access to tools necessary for data workflows.

Metaflow is compatible with Python and R language and any of their libraries, which makes it a very versatile tool. Workflows in Python are defined as subclasses of a class provided by Metaflow, and the steps are defined by creating methods of the class with Metaflow-specific decorators applied.

Workflows in Metaflow can be dynamic based on the data, and the steps of the workflows can create visualizations rendered in the Metaflow UI. Metaflow has a powerful plugin system where additional features required by the developers can be added.

The development of the workflows can be initiated locally, where everything is orchestrated in the local environment. Once there is a need to productionalize the workflows, Metaflow can be deployed to a Kubernetes cluster or AWS cloud, where it can utilize AWS-specific features such as AWS Batch. Once deployed, the workflows can utilize the available resources to scale.



Run ID	Status	User	Started at	Finished at	Duration
332	🟩 Completed	ville	10-12-2021 12:57:59	10-12-2021 12:58:40	41s

Mode: Workflow
Status: All (8) 🟩🟨🟥
Order by: Started at ↓
✓ Group by step

start (1)	2.9s	2.9s
1962	2.1s	2.1s
train_model (5)	8.4s	8.4s
1963	5.9s	5.9s
1964	6.2s	6.2s
1965	5.9s	5.9s
1966	6.2s	6.2s
1967	5.9s	5.9s
join (1)	18.0s	18.0s
1968	17.1s	17.1s
end (1)	3.1s	3.1s
1969	2.1s	2.1s

Figure 2.6: Screenshot of the Metaflow Monitoring UI when deployed [37].

When deployed in Kubernetes, the underlying workflow orchestrator that Metaflow uses can be either Apache Airflow or Argo Workflows, both of which are well-established in the Kubernetes ecosystem. Using Argo Workflows and Argo Events, Metaflow allows for simple workflow scheduling and triggering. [36]

Advantages

+ Simple to use, can be used locally.
+ Built-in version control for pipeline artifacts with deduplication.
+ Efficient scaling with cloud integration.
+ Card visualization for flows.
+ The open-source version offers the same functionality as managed.

Disadvantages

- Lacking administration tools.
- Missing proper user isolation.
- Missing versioning of workflows.

2.3.2.3 ZenML

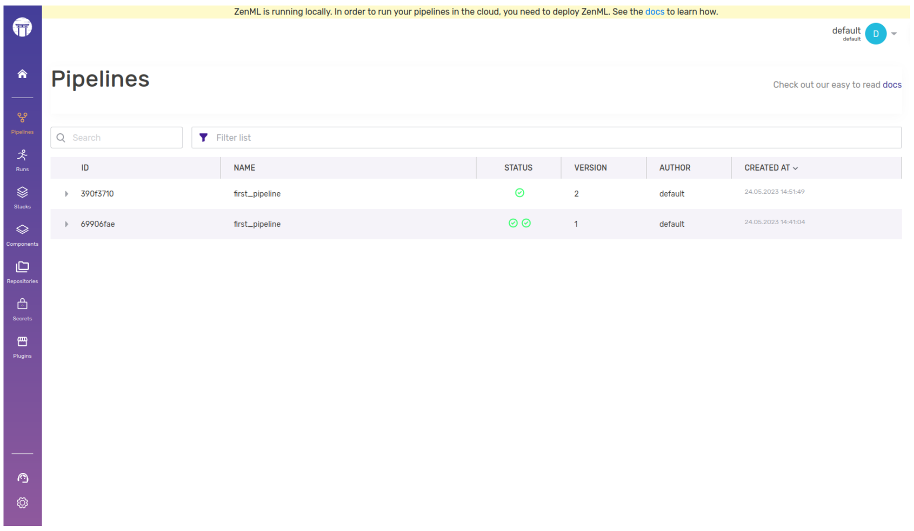
Figure 2.7: ZenML logo [38].

Website	`www.zenml.io`
Creator	ZenML
Maintainer	ZenML
Release Year	2021
License	Apache License, Version 2.0

Table 2.4: The basic information on ZenML [38].

ZenML is one of the newer offerings in the workflow orchestration field. The approach ZenML takes is a bit different from other workflow orchestration tools. It is geared towards being a MLOps platform by integrating the popular tools in the MLOps landscape. This puts it in a spot where it is not an end-to-end MLOps platform but also not just a plain workflow orchestration tool either. It is considered a workflow orchestration tool since its core feature is that it provides a way to define ML workflows.

Its development is highly focused on abstracting the integration of other tools in the MLOps landscape by providing a way to register the tools into the ZenML stack. Currently, over fifty supported tools can be integrated with ZenML and easily used thanks to the abstracted ZenML API in the designed workflows. Custom integration can be created as well to support any tool necessary for the job.


ZenML is running locally. In order to run your pipelines in the cloud, you need to deploy ZenML. See the docs to learn how.

Pipelines

Check out our easy to read docs

ID	NAME	STATUS	VERSION	AUTHOR	CREATED AT
390f3710	first_pipeline	✓	2	default	24.05.2023 14:51:49
69906fae	first_pipeline	✓ ✓	1	default	24.05.2023 14:41:04

Figure 2.8: Screenshot of the ZenML UI [38].

This integration-focused philosophy allows for great extensibility of the tool, providing a way to create multiple stacks using various tools for the outlined step in the ML life cycle. For example, multiple underlying orchestration, artifact tracking, or data validation technologies can be used for the workflows.

ZenML provides tools enabling the reproducibility of ML workflows by automatically tracking and versioning stacks, pipelines, artifacts, and source code.

It is compatible with Python, and its orchestration functionality is provided using decorators from its library. ZenML can be used locally, or the library can be configured to interact with a remote deployment of ZenML when there is a need for scalability.

Most of the ZenML code is open-source, but some of the more advanced features are only available in the managed ZenML Cloud version of the tool[2]. This may be perceived as a threat, where the user may be forced to upgrade based on the missing features.

The great thing about ZenML is that it is infrastructure agnostic, meaning that the underlying deployment infrastructure can be provisioned using any of the leading cloud providers.

ZenML is a promising tool that is becoming increasingly mature with time. New features are being added regularly, which may improve the ML experience of this tool. [38]

Advantages

+ Option to run locally.
+ Infrastructure stack abstraction through unified ZenML API.
+ Great documentation.
+ Integration of the most popular tools.

Disadvantages

- Regular changes to the ZenML API.
- Feature discrepancy between open-source and managed version.

2.3.3 Other Tools

Besides the tools with in-depth overviews, various other tools exist in the space of workflow orchestration and pipelining tools. These tools also provide similar functionalities and may be a better fit depending on the requirements and properties of the ML project. All of the following tools are open-source.

- **DVC (Data Version Control)** is a tool tailored for ML projects. It offers simple workflow orchestration by defining DAG pipelines using YAML to run various commands in steps. The main focus of the tool is data and model versioning to enhance project reproducibility and collaboration. [39]

[2] Comparison between open-source version and ZenML Cloud managed version is available at: `www.zenml.io/open-source-vs-cloud`.

- **Prefect** is a workflow orchestration tool that focuses on simplicity and fault tolerance, providing robust automation and monitoring capabilities for complex data pipelines. [40]
- **Argo Workflows** is a Kubernetes-native workflow engine focused on general workflows. Argo Workflows supports the orchestration of parallel jobs and complex data processing tasks across scalable environments. It is often used as a base workflow engine for ML-specific workflow orchestration tools. [41]
- **Apache Airflow** is designed for creating, scheduling, and monitoring workflows, featuring extensive integration options and a detailed management interface. It is one of the most mature workflow orchestration tools often used as a base workflow engine. [42]

2.4 Experiment Tracking

Experiment tracking tools are essential for the model engineering phase in ML development. They allow proper management of experiments and their artifacts such as parameters, metrics, models, and others [43]. Many times, developers track the results of their experiments manually, writing metrics down into tables. This process is tedious and error-prone.

During the model engineering phase, developers should utilize these tools to log the results of the experiments programmatically and to visualize and compare their results later. Experiment tracking tools enable the developers to draw quick conclusions based on the saved information and easily choose the right model for the task. [44]

2.4.1 Core Features

The following features should be provided by the experiment tracking tools [44, 31]:

- **Metadata logging tools.** Experiment tracking tools should provide mechanisms to log all of the necessary metadata associated with the model training experiment. This metadata can include parameters, metrics, notes, and other artifacts created during the run.
- **Visualization and comparison of experiments.** To compare the logged experiments, these tools should provide ways to visualize the recorded data, allowing for a simple way to compare the models created. In the visualizations, input parameters, resulting metrics, and other metadata associated with the experiment can be compared.
- **Model versioning and lineage.** Not all experiment tracking tools provide a model registry component, but it is certainly a valuable feature to have included in the experiment tracking tool. That way, experiments can have their resulting models associated with them, and a final trained model can be easily chosen based on the experiment visualizations in the tool.

- **Integrations.** It is crucial that experiment tracking tools allow for integration with popular ML libraries. These integrations often enhance the experience of using these tools by allowing automatic logging without the need to log every artifact from the run manually. It is also important that the experiment tracking tool can be used inside orchestrated workflows and with CI/CD tools.

2.4.2 Available Tools

In the open-source space, there are not many experiment tracking tools available. Most of the tools in the MLOps landscape provide managed versions of their tools to be profitable and to support development. When using managed tools, the maintenance complexity is much lower, and it might be worth considering paying for such tools. However, open-source tools still exist in this space, and we chose the following tools for a more in-depth overview.

2.4.2.1 Aim

Figure 2.9: Aim logo [45].

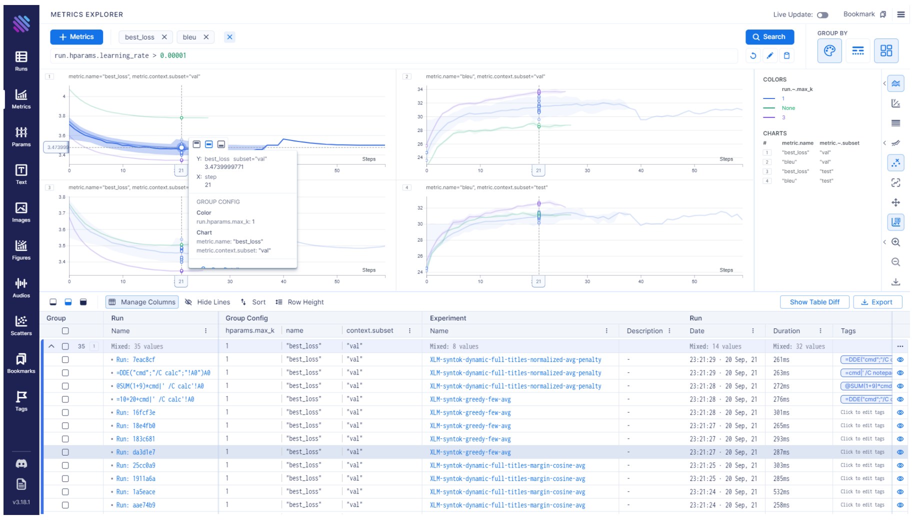
Website	`www.aimstack.io`
Creator	AimStack
Maintainer	AimStack
Release Year	2020
License	Apache License, Version 2.0

Table 2.5: The basic information on Aim [45].

Aim is an open-source experiment tracking tool providing visualizations of complex metadata to allow for quick comparison of experiments. Its powerful UI enables the developers to compare a variety of metadata types such as metrics, parameters, text, images, and others. Another feature of the UI is a complex query language that makes it possible to view experiments based on a set of requirements.

An overview of the logged metadata can be done through each run separately, or there are tabs in the UI based on the metadata type. In each of these tabs, the developer can create a visualization, helping them to get an idea of the experiments' results. These visualizations can be bookmarked for quick access later. Data can also be accessed through Aim's API, which is abstracted by its client library.

The main functionality is provided as a Python library to be integrated into model experiment runs to enable logging. Aim's library integrates multiple popular ML tools to enable logging metrics automatically without much manual logging code. Aim can be used locally or deployed as a remote tracking server to enable collaboration on ML projects.



```
run.hparams.learning_rate > 0.00001
```

Group	Run Name	hparams.max_k	name	context.subset	Experiment Name	Description	Date	Duration	Tags
35	Mixed: 35 values	1	"best_loss"	"val"	Mixed: 8 values		Mixed: 14 values	Mixed: 32 values	
	Run: 7eac8cf	1	"best_loss"	"val"	XLM-syntok-dynamic-full-titles-normalized-avg-penalty	-	23:21:29 · 20 Sep, 21	261ms	=DDE("cmd";"/C c
	=DDE("cmd";"/C calc";"!A0")A0	1	"best_loss"	"val"	XLM-syntok-dynamic-full-titles-normalized-avg-penalty	-	23:21:29 · 20 Sep, 21	263ms	=cmd|' /C notepa
	@SUM(1+9)*cmd|' /C calc'!A0	1	"best_loss"	"val"	XLM-syntok-dynamic-full-titles-normalized-avg-penalty	-	23:21:28 · 20 Sep, 21	272ms	@SUM(1+9)*cmd
	=10+20+cmd|' /C calc'!A0	1	"best_loss"	"val"	XLM-syntok-greedy-few-avg	-	23:21:28 · 20 Sep, 21	276ms	=DDE("cmd";"/C c
	Run: 16fcf3e	1	"best_loss"	"val"	XLM-syntok-greedy-few-avg	-	23:21:28 · 20 Sep, 21	301ms	Click to edit tags
	Run: 18e4fb0	1	"best_loss"	"val"	XLM-syntok-greedy-few-avg	-	23:21:27 · 20 Sep, 21	265ms	Click to edit tags
	Run: 183c681	1	"best_loss"	"val"	XLM-syntok-greedy-few-avg	-	23:21:27 · 20 Sep, 21	293ms	Click to edit tags
	Run: da3d1e7	1	"best_loss"	"val"	XLM-syntok-greedy-few-avg	-	23:21:27 · 20 Sep, 21	287ms	Click to edit tags
	Run: 25cc0a9	1	"best_loss"	"val"	XLM-syntok-dynamic-full-titles-margin-cosine-avg	-	23:21:25 · 20 Sep, 21	303ms	Click to edit tags
	Run: 1911a6a	1	"best_loss"	"val"	XLM-syntok-dynamic-full-titles-margin-cosine-avg	-	23:21:25 · 20 Sep, 21	285ms	Click to edit tags
	Run: 1a5eace	1	"best_loss"	"val"	XLM-syntok-dynamic-full-titles-margin-cosine-avg	-	23:21:24 · 20 Sep, 21	532ms	Click to edit tags
	Run: aae74b9	1	"best_loss"	"val"	XLM-syntok-dynamic-full-titles-margin-cosine-avg	-	23:21:24 · 20 Sep, 21	258ms	Click to edit tags

Figure 2.10: Screenshot of the Aim UI.

Aim's powerful UI can also be integrated with other backends, not just its own. It can be used, for example, with MLflow or spaCy backends. [45]

Advantages

+ Integrations with popular libraries.
+ Powerful visualization UI.
+ Various metadata type comparisons.
+ Can be run directly in Jupyter notebooks.

Disadvantages

- Complicated self-hosting.
- Missing user isolation.

2.4.2.2 MLflow

Figure 2.11: MLflow logo [46].

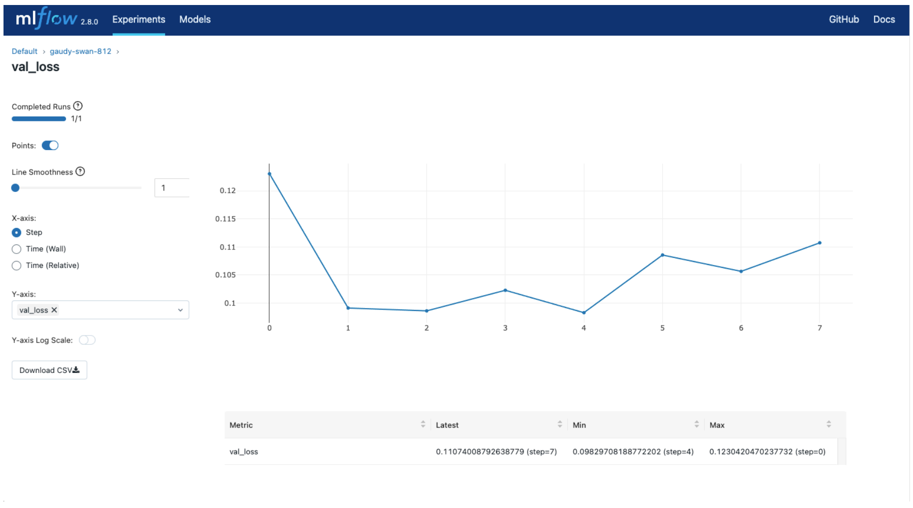
Website	`www.mlflow.org`
Creator	Databricks
Maintainer	Databricks
Release Year	2018
License	Apache License, Version 2.0

Table 2.6: The basic information on MLflow [46].

MLflow is an open-source ML platform with a primary focus on experiment tracking. It provides multiple components that are helpful along the ML life cycle, including metadata tracking, model registry, and model deployment components. This makes it versatile in the MLOps pipeline since it can replace multiple services.

MLflow tracking allows the users to track ML experiments and log respective metadata such as parameters, metrics, dataset information, and other artifacts. The model registry is closely tied with experiments, where each experiment run can have a model logged to the registry as its output.

MLflow's UI allows a quick preview of past experiments and their respective runs. In the experiment preview, the developers can compare metrics, which enables them to choose the best model. All of the resources in the MLflow deployment can be accessed through API, allowing data to be retrieved programmatically.



val_loss

Completed Runs 1/1

Points:

Line Smoothness: 1

X-axis: [✓ Step] Time (Wall) Time (Relative)

Y-axis: val_loss

Y-axis Log Scale:

Metric	Latest	Min	Max
val_loss	0.11074008792638779 (step=7)	0.09829708188772202 (step=4)	0.1230420470237732 (step=0)

Figure 2.12: Screenshot of the MLflow UI [46].

MLflow can be used locally or deployed as a remote tracking server. Remote deployment allows for collaboration where the developers can compare ML experiment runs. When deployed, the model registry component can be used to retrieve tagged models for production.

Since MLflow has become very popular in the MLOps space, it is compatible with a variety of tools and libraries. New integrations and plugins are still being created. [46]

Advantages

+ Includes extra functionality.
+ Popular in the MLOps landscape.
+ API for multiple languages.
+ Well documented.
+ Good compatibility.

Disadvantages

- Limited scalability.
- Missing user isolation.

2.4.3 Other Tools

There are other tools available in the experiment tracking landscape. Except for MLflow, it seems that the most popular solutions for experiment tracking are the managed ones. The reason for this might be that most of the managed tools are free for individual and academic use, which makes it compelling to use since there is a simple setup process associated with these tools. Other open-source tools which also provide managed experience are:

- As mentioned before, **DVC (Data Version Control)** includes multiple components helpful in the ML process. One of the components included is the experiment tracking tool. [39] Visualizations provided by this tool are a little less powerful compared to the applications mentioned before.
- **Sacred** offers experiment tracking but does not allow for collaboration since it is available only locally. The front end is not included by default, but there are community efforts to provide UI for the metadata stored by this tool. [47]

2.5 Other Categories

In the past few sections, we have presented the crucial tools for implementing the pipeline. The exact MLOps pipeline needs to be adjusted according to the project requirements. There are other tools that may be beneficial or even necessary for some projects, which is why we will give a short overview of additional tools in the MLOps landscape.

2.5.1 Data Storage and Versioning

As mentioned in the chapter 1 describing machine learning itself, data is one of the crucial ingredients in ML project development. However, handling data in ML projects can become problematic since large amounts of data are often needed to train ML models properly. Data storage and versioning tools try to solve these problems by providing a way to store, version, and distribute larger amounts of data. This allows for collaboration, versioning, and reproducibility, as pointed out in the MLOps principles.

Most of the tools in this space provide Git-like versioning features that allow for the easy storage, loading, and versioning of datasets. Some of the open-source tools include:

- **DVC (Data Version Control)** extends version control systems to handle large data files. It tracks versions using lightweight metafiles while the actual data is stored in a remote repository. DVC integrates seamlessly with existing Git workflows. DVC also provides other tools that are helpful in the ML life cycle, as mentioned before. [39]
- **Git LFS** enhances Git by allowing users to manage large files without bloating their repositories. It stores references to large files in Git, while the files themselves are hosted on a separate server. [48]
- **Pachyderm** is not only a data version and storage tool, but it also provides data-driven pipeline orchestration. Data versioning is done in a manner similar to a Git workflow, where there are commits, branches, and other familiar concepts. [49]

2.5.2 Feature Stores

Feature stores act as a central hub for feature data and metadata in ML projects. They enable collaboration and reusability of features extracted by data engineers, reducing duplication of data engineering efforts. [50]

- **Feast** is an open-source customizable operational data system to manage and serve machine learning features. [51]
- **Hopsworks** is a modular data platform. It can be used as a standalone feature store, and it also includes other functionalities that enable the development and operation of feature pipelines. [52]

2.5.3 Hyperparameter Optimization

Hyperparameter optimization tools can be used to get the best set of parameters for a model on a particular dataset. By using these tools, the developer can avoid the tedious manual work of setting the parameters manually. [53]

- **Optuna** is an open-source framework for hyperparameter optimization in ML. It implements a variety of algorithms to search hyperparameter space and find optimal values. Optuna also provides a dashboard to get a real-time preview of tuning. [54]

- **Ray Tune** is an open-source Python library for experiment execution and hyperparameter tuning. Ray Tune excels in scaling the tuning process by utilizing parallelization across multiple nodes without any changes to code. [55]

2.5.4 Model Registry

The model registry component provides a centralized repository to enable effective model management and documentation. Models and their associated metadata are stored together. Having a model registry helps with collaboration and versioning.

- **MLflow**, described in the subsection 2.4.2.2, also includes a model registry component. Having the model registry in the experiment tracking tool allows us to find the exact experiment and metadata associated with the model. [46]
- **Verta Model DB** is an open-source model registry component of the Verta MLOps platform. Its model registry can be used separately, but usage of the whole platform may be beneficial. [56] The development of this project is currently not very active.

2.5.5 Model Deployment and Serving

Created models need to be deployed into production. There are many ways to deploy a model into production. These tools should provide automated scaling and load balancing across multiple instances of deployed models. Batch processing and real-time inference should be supported as deployment patterns. [31]

- **BentoML** is an open-source model serving framework that provides a unified format to package various ML models into deployments. This package can then be built into a Docker image to be deployed to Docker-compatible platforms. [57]
- **MLflow**, described in the subsection 2.4.2.2, also provides model deployment and serving tools among its many features. Models registered into its registry can be easily served locally for testing. Then, they can be packaged as a Docker image, which can then be deployed to Docker-compatible platforms. This Docker image exposes an API where the inference requests are processed. [46]

2.5.6 Model Monitoring

To get insights after the ML model has been deployed into production, model monitoring platforms should allow the developers to detect issues such as data or concept drift. According to the monitoring results, steps should be taken to fix detected issues. [17]

- **Deepchecks** is an open-source solution including multiple components to test, monitor, and evaluate the model's performance across its entire development process. It provides a beautiful dashboard to preview gathered insights. The downside is that the open-source version is limited to monitoring only one model per deployment. Deepchecks cannot be used locally. [58]

- **Evidently AI** comes with an extensive library of reports, test suites, and a monitoring dashboard. Evidently AI can be used locally, or the dashboard can be deployed as an external service. Evidently AI is still in a pretty early development, which means features are being added very often. [59]

Chapter 3

Provided Application

In this chapter, we will analyze the application that uses machine learning provided by Profinit EU. First, we will introduce the application while redacting its primary purpose and implementation details since it is Profinit's intellectual property. Next, there will be a high-level overview of the architecture and the technologies used. Since the goal of this thesis is not the application development but rather the use of general principles applicable universally, we should not look at the code architecture in great detail. Lastly, we will look at the current deployment strategy.

3.1 Overview

The provided application is available in the form of a web application that utilizes a great amount of processed data to achieve its main business task. It has yet to be provided to customers. The development began in April 2021 internally with the intention to sell the application to customers if it successfully solves their business problems. The application is moderately complex, and since its development sprung off of an Minimum Viable Product (MVP), its architecture evolved iteratively.

Nowadays, the application is becoming more mature and production-ready. Large amounts of data and leveraging machine learning make the life cycle of this application considerably more complex than other applications since the development of the model is technically separate from the development of the web application.

3.1.1 Application Architecture

Components of the application are displayed in the diagram 3.1, which describes the high-level architecture.

The user interacts with the web application instance using an Application Programming Interface (API) or Web User Interface (UI). They can make a query that gets sent from the web application using a message broker to a task queue for processing. This enables asynchronous processing of lengthier tasks, which would otherwise block user interaction for some time. If the query is done using an API, the user is given a task ID, and the task is processed asynchronously. Otherwise, the task gets awaited if it is done through

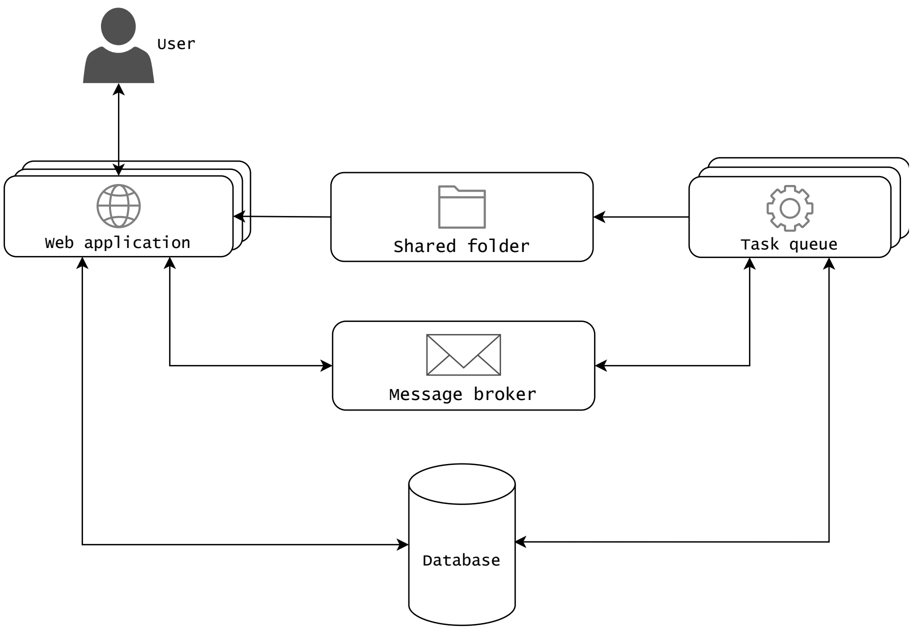


Figure 3.1: High-level diagram of architecture of the provided application.

Web UI. The task queue worker updates information about the task's status, and the task result is passed to the web application using a shared folder between the web application and the task queue. Users can retrieve the result of a task by making requests about the task's status and requesting its result when the task is finished processing. Data for the task processing and other information are stored in the database.

3.1.2 Technologies Used

Since the application utilizes machine learning, it may come as no surprise that the language used to implement both the task queue and the web application is Python. Exact technologies used to provide the functionality are shown in a diagram 3.2. Below is an overview of each of the technologies used in the implementation.

- **Python.** *"Python is an interpreted, object-oriented, high-level programming language with dynamic semantics."* [60] Thanks to its large ecosystem of libraries and modules focused on data science, it is an excellent choice for applications utilizing machine learning.

- **Flask.** Flask is a micro web framework written in Python that provides configuration and conventions with sensible defaults. Since it is a micro web framework, it does not include any database abstraction layer, form validation, or other tools out of the box, but it provides support for extensions to add such features. [61] Using Flask's client-side session enabled by storing cookies in the client's browser, the web application instance is stateless.

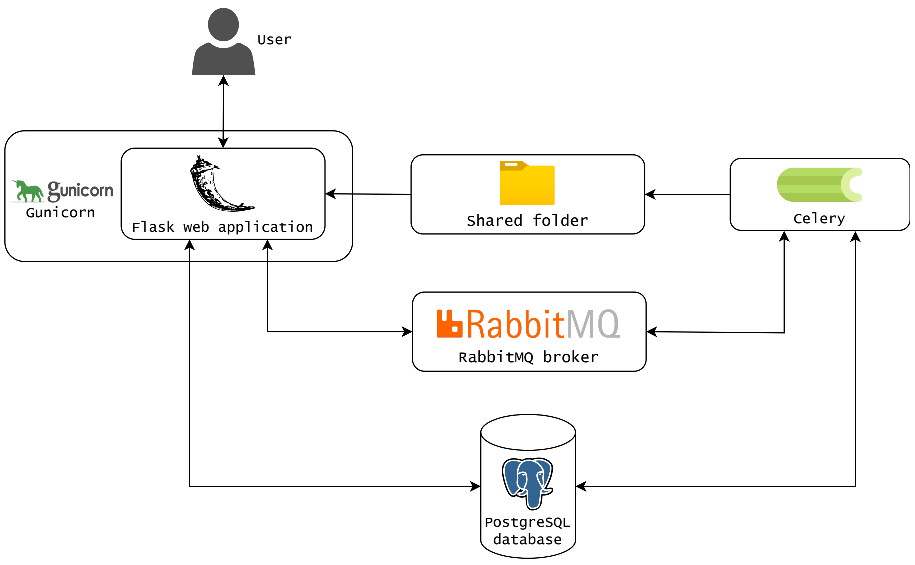


Figure 3.2: High-level diagram displaying the technologies used for each component of the application.

- **Gunicorn.** *"Gunicorn 'Green Unicorn' is a Python WSGI HTTP Server for UNIX*[3]*. It's a pre-fork worker model. The Gunicorn server is broadly compatible with various web frameworks, simply implemented, light on server resources, and fairly speedy."* [62] This server is used to serve the Flask application to clients.

- **Celery.** Celery is an open-source distributed task queue library written in Python [63]. It processes tasks created by the users using the Flask web application. Processing can be done asynchronously, or the web application can await the tasks. The results of the tasks are stored in the database, and the resulting assets are shared with the web application using a shared folder between the web application and the workers.

- **RabbitMQ.** RabbitMQ is an open-source messaging and streaming broker [64]. This technology provides communication between the web application and the Celery workers of the task queue.

- **PostgreSQL.** PostgreSQL is an open-source object-relational database system that uses and extends the SQL language [65].

3.1.3 Use of Machine Learning

For our assignment, it is necessary to understand how machine learning is utilized in this application.

By analyzing the source code and communicating with the developers, we have learned that the application uses the machine learning model to calculate a metric based on features extracted from the initial dataset. This metric calculation is done in advance using batch inference before the application is

[3] Unix is a family of multitasking, multi-user computer operating systems.

deployed, and the results are saved in the database. When answering the user's queries, data is retrieved from the database and returned to the user.

The model is also used during the operation of the application, but only when answering user queries done through the Web UI, to calculate SHapley Additive exPlanations (SHAP)[4] values explaining the result of the query based on the input features.

The application uses the model by packaging it and including it in the task queue. This was done because the model size is less than two megabytes, and the inclusion, as opposed to the separate deployment of the model, seemed more straightforward to the application developers.

3.2 Development

Only a few developers were working on this solution during the application's development. Thanks to this fact, the collaboration on the development was pretty straightforward, with the developers communicating directly with each other.

Everything regarding the application is kept in the same repository, including the initial data, analysis notebooks, tasks code, and web application code. From the information obtained by discussing with the application's developers, we have learned that the application is developed on a Virtual Machine (VM) exposed to the developers.

3.2.1 Development Process

The development process of this application can be split into parts, considering the necessary steps taken from the conception to the current state of the application's components.

- **Data Engineering.** When the development started, a folder was created on the development VM in which a Git repository was initialized. The obtained unprocessed data was not versioned into the repository since it is over three gigabytes in size.

  The data was later processed and analyzed, resulting in features used by the model to predict the result. Transformed data in the final state was versioned using Git Large File Storage (LFS). One of the reasons for versioning this data was that it is required to set up the application.

  Data gathering for the purposes of the application is done with the help of scripts that are run manually since the data does not dynamically change. The sources for the gathered data are primarily static, with the decisions of the inclusion of new sources of data related to the business problem at hand being made manually.

- **Model Development.** Based on the features computed from the data, an analysis was performed concerning model choice and hyperparameter tuning. This helped the model achieve the optimal results when calculating the resulting metric based on the input features.

[4] See paper [66].

Currently, the model is not continuously developed since the application data stays mostly the same. When there is a decision to include new data related to the business problem, the model is refined based on the result of an analysis of the newly included data.

- **Web Application Development.** After developing the model, the web application was created to expose the data obtained from the analysis to the potential users of the application using API or Web UI.

  Minor changes to the model do not necessitate changes to the application. However, including new data sources may lead to changes to the web application since the relevant data to the result of the query is displayed to the user.

- **Task Queue Development.** Alongside the web application development, code for tasks to be executed in the task queue was developed. As mentioned before in the section 3.1, the web application depends on the task queue to process longer-running tasks.

  The packaged model is included in this application component since it is used to explain the results of the user query when querying the application using the Web UI.

- **Application Testing.** Developers of the application created tests. These tests were automated using GitLab CI/CD pipelines. The tests build a test image against which end-to-end and integration testing is performed.

3.3 Deployment

The application is currently deployed to a VM with publicly exposed HTTP endpoints accessible by the users[5]. It is packaged into a Docker container image and orchestrated using Docker Compose.

The deployment consists of four Docker containers. One is for the web application, another for the Celery worker, and the last two are for the PostgreSQL database and RabbitMQ message broker. These containers communicate using the Docker network. The shared folder between the web application and the Celery worker is provided by Docker volume and stored on the VM file system. Another Docker volume is provisioned on the file system for the data of the PostgreSQL database. The deployment diagram of the provided application is depicted in diagram 3.3.

[5] Endpoints may be exposed using reverse-proxy application.

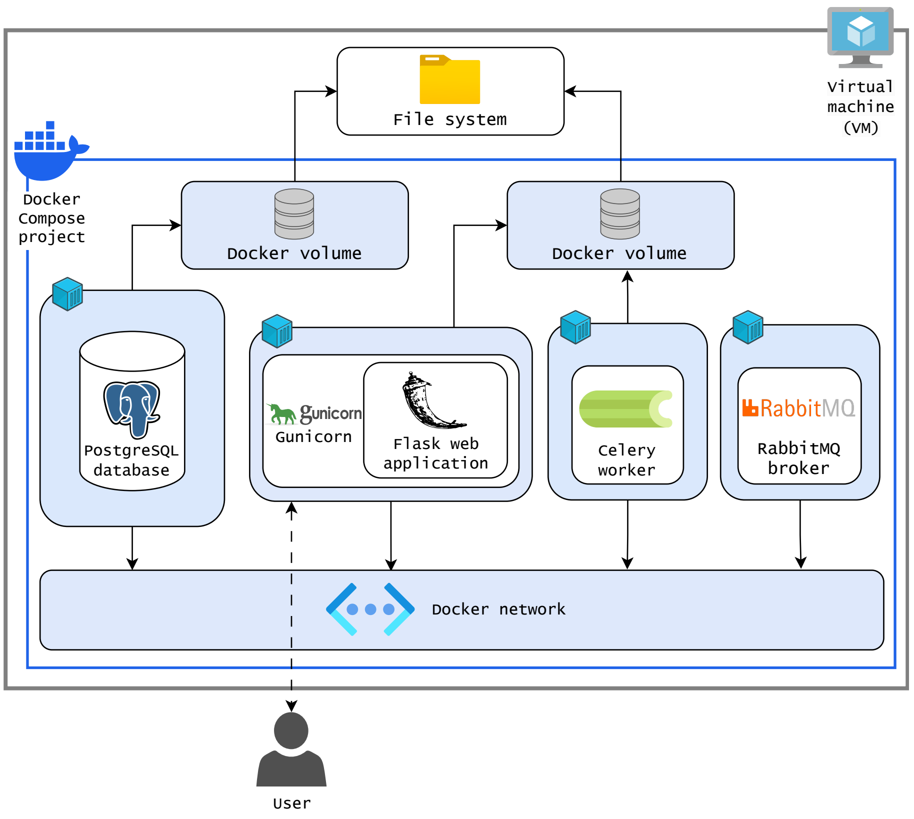


Figure 3.3: Deployment diagram of the application.

3.3.1 Deployment Process

The application is deployed manually, taking a series of steps to get the necessary data, train the model, create assets, and package it all into the Docker image. This Docker image is then run as the application in a container orchestrated by Docker Compose. The deployment process can be summarized into the following steps:

1. Connect to the VM using SSH.
2. Pull the changes from the remote GitLab repository.
3. Run the orchestration scripts for the creation of the application's static assets.
4. Train the model with the new data and predict the values of the metric.
5. Rebuild the image for the web application and worker[6].
6. Swap the currently running containers of web application and worker.
7. Fill the database with the newly obtained and orchestrated data.

[6] Worker and the web application run using the same image, using different entrypoint.

This process is also illustrated in a diagram 3.4. After successfully completing all of these steps in order, the new version of the application with the new model and data is deployed on the VM.

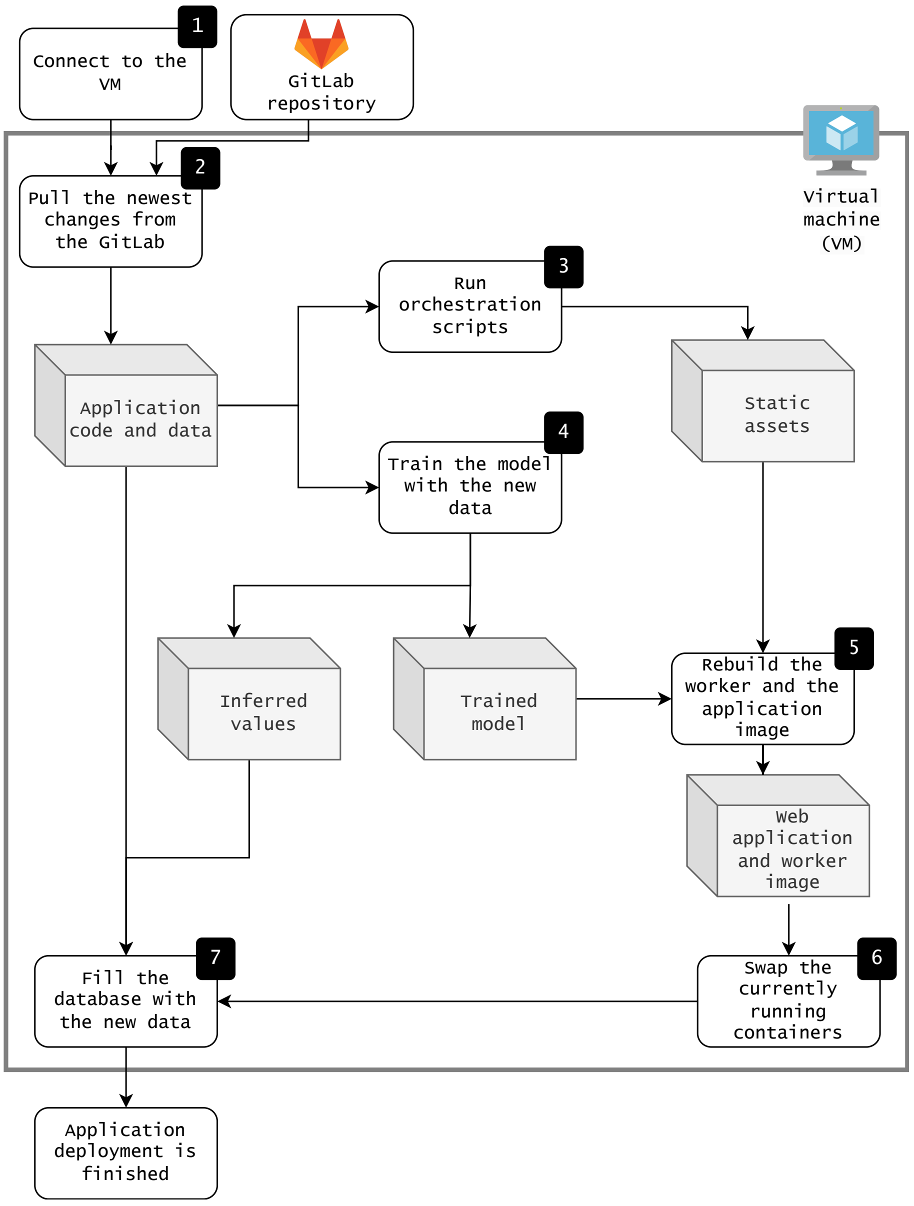

1
Connect to the VM
GitLab repository
Virtual machine (VM)
2
Pull the newest changes from the GitLab
3
Run orchestration scripts
Application code and data
Static assets
4
Train the model with the new data
Inferred values
Trained model
5
Rebuild the worker and the application image
Web application and worker image
7
Fill the database with the new data
6
Swap the currently running containers
Application deployment is finished

Figure 3.4: Deployment process diagram of the application with depicted manual steps that must be taken to deploy the new version of the application.

CHAPTER **4**

Pipeline Design

This chapter will describe an extensible and portable ML pipeline design. This design will be later utilized as the base for the pipeline implementation for the provided application. First, we will describe the requirements that we posed that should be satisfied by the pipeline. Later, we will introduce the designed pipeline and its components. Lastly, we will provide reasons for the tools chosen in the pipeline.

4.1 Design Considerations

The preceding chapters provided an overview of MLOps, the tools available to implement its principles, and the application provided by Profinit EU. From the description of the provided application, it is clear that no resource-heavy and complex machine learning is being done, so it is worth considering whether implementing a pipeline using any previously described tools is even necessary. Considering the experimental nature of this thesis, described in the introduction, we have decided to implement the pipeline but with a focus on project portability and extensibility.

4.2 Requirements

To get a concrete description of what we want to achieve with our implementation of the MLOps pipeline, we have created a set of requirements to be met by the final design. Requirements for the pipeline are divided into two distinct groups: functional and non-functional requirements.

4.2.1 Functional Requirements

Functional requirements define the system features that need to be implemented. To get a solid grasp on the MLOps principles and tools, we have established the following requirements:

- **FR1: Workflow orchestration**
  The pipeline should enable the orchestration of workflows facilitated by defining steps that will be taken when the workflow is executed. Steps

in workflow orchestration should have the ability to branch out dynamically based on the data at runtime.

- **FR2: Experiment tracking**
Developer should have the ability to compare different experiments performance preferably in an easy-to-use and accessible UI.

- **FR3: Model registry**
Models should be saved to a model registry, where they can be downloaded and preferably tagged based on their characteristics and utilization (for example, whether the model is ready for production, needs evaluation, or shows a specific behavior to be analyzed).

- **FR4: Deployment monitoring**
Monitoring of the deployment should be enabled by the pipeline so that when the performance degrades, necessary steps can be taken or automated based on the insights from monitoring. Monitoring insights should be available programmatically, and a UI should also be available.

- **FR5: Data versioning**
Data necessary for the ML project should be versioned to get the exact data used for the model training.

4.2.2 Non-Functional Requirements

Non-functional requirements dictate the characteristics that the implemented system should pose. Since this is an experimental implementation, we have concluded the following non-functional characteristics that focus on future improvements:

- **NFR1: Extensibility**
Pipeline design should be extensible. It should be possible to add additional features and make improvements after it has already been implemented.

- **NFR2: Project portability**
The pipeline design should be created in a way that is not specific to any particular project in which it may be utilized. In some projects, changes may be necessary, but the divergence from the base pipeline design should be minimal.

- **NFR3: Scalability**
Scalability with the growing resource demand should be possible. It also applies in reverse; when the resources are not utilized, they should be downscaled to a minimum.

- **NFR4: Maintainability**
It should not be difficult to update the pipeline or change parts of the underlying infrastructure.

- **NFR5: Recoverability**
When some pipeline steps fail, they should not pose a risk to systems availability, and there should be a way to recover from a failure.

- **NFR6: Infrastructure vendor independence**
  With minor changes, the pipeline should be implementable on an infrastructure provided by another vendor or an on-premise infrastructure.

4.3 Pipeline Overview

The final design of the pipeline is very complex, utilizing multiple tools in a highly interconnected manner to try and fulfill all of the requirements set beforehand. Due to the complexity, here we provide only a simplified overview of the components of the designed pipeline. The tool providing the CI/CD is left out of the diagram since it and its jobs should be chosen based on the implementation requirements.

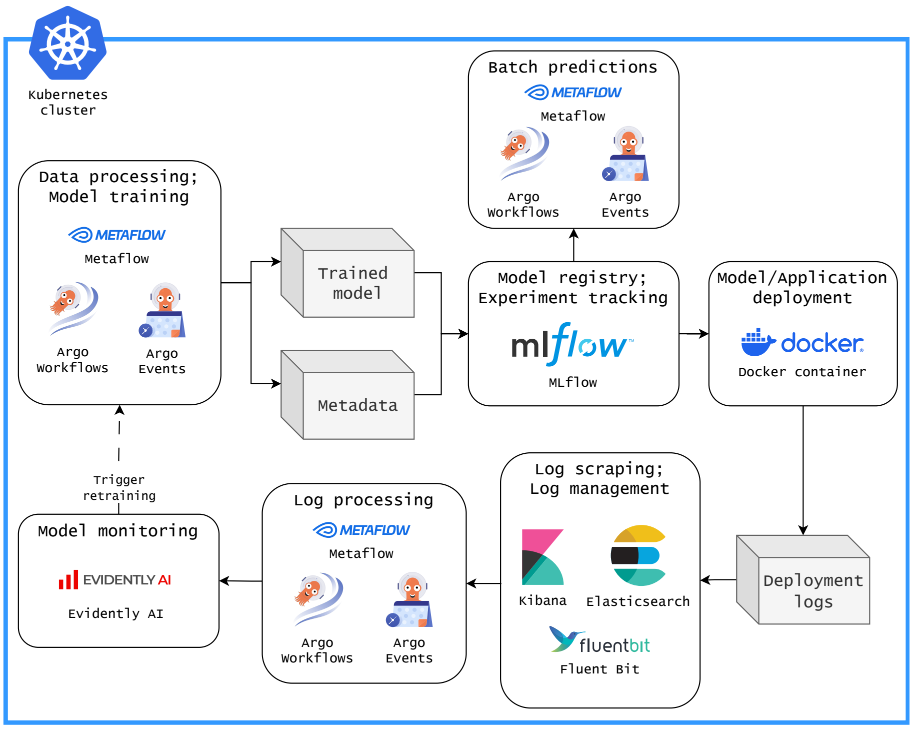


Figure 4.1: Diagram of the pipeline with details omitted, capturing the main components of the pipeline

4.3.1 Process Description

The pipeline starts with a data processing component utilizing Metaflow with Argo Workflows as an orchestrator. This part provides the workflow orchestration capabilities to implement steps of the data engineering phase. Metaflow also provides a data versioning feature. The processing can be initiated manually or based on an event using Argo Events.

After processing the data, the model training workflow is implemented using Metaflow and Argo Workflows. One of the outputs of this step – the metadata used to track the experiment is pushed to MLflow. MLflow also acts as a model registry component, enabling the versioning of models and comparing their metrics. The trained model is pushed to the model registry.

When the models and their experiments are compared, and the best is selected, batch predictions can be made utilizing Metaflow and Argo Workflows. Suppose the project does not utilize batch predictions, and the model must be deployed in an application as a dependency or inference service. In that case, the developer should choose a way to pack the model or the application into a Docker image and deploy it on the cluster.

The model or application deployment should generate usage logs, which can then be scraped using Fluent Bit. These logs are then pushed into the Elasticsearch database. Logs in the database can be previewed using Kibana.

These logs are processed using scheduled runs of Metaflow workflows using Argo Workflows scheduling capabilities. After processing the logs, reports and tests can be created using the Evidently AI monitoring tool and are pushed into the Evidently AI monitoring UI.

Based on the results of tests or outputs of the reports created using Evidently AI, the developer can trigger retraining, or it can be triggered automatically using the Argo Events webhook feature.

4.4 Component Overview

Given the requirement for scalability **[NFR3]** and to broaden the knowledge of modern technologies, Kubernetes is chosen as the base system for the pipeline.

Another benefit of using this technology is that we satisfy the requirement of infrastructure vendor independence **[NFR6]** since Kubernetes can be installed manually on-premise, in the cloud, or is provided as a service by most of the big cloud providers such as Amazon Web Services EKS, Azure AKS, Google Cloud GKE, and many others…

4.4.1 Data Processing

The data processing component is facilitated by Metaflow deployed with the Argo Workflows orchestrator. This technology allows orchestrating workflows using steps to define a DAG satisfying **[FR1]**.

Deployed on Kubernetes, Metaflow also allows to scale based on the resource requirements set by the developer **[NFR3]**.

In the Metaflow framework, there is also an ability to version artifacts created in the workflow; this can be used to version the data **[FR5]**.

Data processing can be initiated on schedule, manually, or using any of the event sources compatible with Argo Events. Multiple triggering options provide flexibility when considering different needs for data processing based on a project requirements **[NFR2]**.

When some step of the workflow fails, Metaflow allows the restart of a workflow run from the failed step with a new fixed pipeline published **[NFR5]**.

4.4.2 Model Training

The model training component uses the same tools as the data processing component. Using the same tools means that it offers the same features, which are applicable and beneficial for model training.

Metaflow poses no limitations on the compatibility of libraries used since Metaflow API in scripts is provided as a Python library. This allows the developers to use a variety of different tools to create, train, tune, and perform a variety of other tasks with their models in their Python scripts **[NFR1, NFR2]**.

Metaflow does not have to be used only in the pipeline. It can execute separate workflows for each user, allowing the developers to experiment with more resources available than in their local environment and check whether the workflow runs correctly in the remote environment.

4.4.3 Experiment Tracking

MLflow is used as an experiment tracking tool **[FR2]**. It allows the model training workflow to save metadata about the executed training runs. Metadata can include dataset information, parameters, metrics, tags, and artifacts. Thanks to the UI exposed by the MLflow service, developers can quickly get an overview of experiments and compare the results.

MLflow supports most of the commonly used ML libraries and frameworks. However, if the developers opt for some not widely used or custom solution, they can use it with MLflow after following steps in its documentation **[NFR1]**.

4.4.4 Model Registry

MLflow includes a model registry component in its features. In collaboration with the experiment tracking feature, this component provides model lineage (information about the experiment that produced the model), model versioning, model aliasing, tagging (used to store information on which Metaflow workflow trained the model), and annotations **[FR3]**.

As mentioned in the section 1.4, versioning is one of the key principles to properly implementing the MLOps paradigm, and this component provides the versioning of the model part of the ML process.

The aliasing feature of the MLflow model registry is used in this pipeline to label the model with the best performance, which can then be downloaded during the deployment step using the alias. When running the training experiment, the registered model can be automatically aliased, or the alias can be applied manually after the experiment has finished.

4.4.5 Model Deployment

Since there are various possible deployment strategies for ML models, this part of the pipeline design is unopinionated. The preferred way of deployment is using a Docker container, which provides a portable packaged environment. There are multiple solutions providing model-serving capabilities that can be used with this setup.

If necessary, as is the case with the provided application, it is possible to deploy not just the ML model as a service but the whole application, with the model included. The deployment strategy depends very much on the exact architecture of the application utilizing ML.

Our pipeline only requires the deployment to generate logs of interactions with the model, enabling us to monitor the model's performance.

Our pipeline also allows batch inference using workflow orchestration provided by Metaflow and Argo Workflows. Using these tools comes with all of the benefits mentioned above. For example, the workflow can be triggered on schedule or using a webhook with optionally provided parameters. Batch inference results can be saved as Metaflow artifacts, or there are other options like uploading them to object storage, for example.

4.4.6 Monitoring

Monitoring is designed in a non-invasive way, where the developers do not need to make many adjustments to the model/application except to log the interactions with the ML model using standard output logs.

The EFK (Elasticsearch, Fluent Bit, Kibana) monitoring stack is selected to aggregate logs generated by the deployment. Each component of the EFK stack has a specific function in the log aggregation and management process. Elasticsearch is used to store and index logs. Fluent Bit collects the logs from the deployment, enriches them with additional information, and ingests them into Elasticsearch. Kibana allows the developers to visualize logs and interact with the data stored in Elasticsearch. This stack is not ML-specific; it is widely used in the industry.

To get ML-specific metrics and monitoring, we utilize this stack with the help of the Evidently AI ML monitoring tool. Using the Argo Workflows workflow scheduling feature, we can schedule the monitoring based on our need to run at regular intervals. During the workflow run, the data is loaded from the Elasticsearch database and preprocessed. Then, the original data used to train the deployed model can be loaded from the Metaflow workflow (which trained the deployed model) and used to generate reports and run tests using Evidently AI. Results of the reports and tests are then pushed into Evidently AI monitoring dashboard **[FR4]**.

Insights gained from this process can be then used to trigger training workflow automatically (using Argo Events webhook) or manually by the developer after viewing the results in the monitoring dashboard. This completes the feedback loop where the model needs to be retrained based on its performance when deployed in production.

4.5 Tool Selection

The exact tool selection is based mainly on the experience of developers at Profinit EU. The tools described in the previous chapter have a similar feature set, making selection of the right tools complicated without long-term evaluation of each tool in practice. Given the thesis's limited timeframe, we have settled on the abovementioned tools after a short trial of the overviewed tools in chapter 2.

The selected tools are easily replaceable by another candidate tool, and the pipeline is modular. The separation of concerns of each tool enables maintainability **[NFR4]**. Since the tools are separate, they do not strongly depend on the versions of other tools in the pipeline, making the upgrades more straightforward as well.

4.5.1 Workflow Orchestration and Pipelineing

Metaflow with Argo Workflows as the orchestration backend is chosen as the workflow orchestration and pipelineing tool. This tool is selected thanks to its flexibility and extensive feature set of the open-source version, which is the same as with the managed version. Another advantage is the automatic tracking of the metrics and pipeline running information and a simple setup for local experimentation and development. In addition to Metaflow and Argo Workflows, Argo Events is deployed to trigger workflows based on various inputs and events.

4.5.2 Experiment Tracker

MLflow is the tool of choice when considering experiment tracking. It was chosen because it is a versatile tool providing valuable components for ML projects. Its integration of the model registry with experiment tracking proved helpful in picking the best model based on the visualizations in the MLflow UI. MLflow can also be used locally without the need for external deployment. Another reason is its popularity. The tool has a large user base, detailed documentation, and many available tutorials.

4.5.3 Monitoring Component

For monitoring, Evidently AI was chosen because its open-source version has all of the necessary features available without any limitations compared to the managed version. Similar to other tools chosen in the pipeline, Evidently AI can be used locally. This tool is still in its early development, but it has already become very popular.

Chapter 5

Pipeline Implementation and Application Migration

This chapter focuses on the actual implementation of the experimental pipeline for the provided application. First, the underlying infrastructure with the tools used for deployment is described. Next, there is an overview of the resources deployed for the pipeline implementation. After that, the migration of the application to the Kubernetes cluster is described. Lastly, there is a look at CI/CD automation used in the pipeline and a summary to paint a picture of the process when using the pipeline.

5.1 Infrastructure

Infrastructure is at the foundation of the pipeline and application deployment. For this experimental instance of the pipeline, a cloud computing based infrastructure with Kubernetes as the underlying system for the pipeline is chosen based on the pipeline's and tools' requirements.

5.1.1 Cloud Computing

Cloud computing or "the cloud"[7] provides users with on-demand resources and computing power. Many companies are moving to the cloud since it provides excellent scalability and lowers maintenance costs, and the companies pay only for the resources they have used [68].

The cloud computing platform Microsoft Azure has been chosen as the infrastructure provider for this project. This choice technically has no implications on the pipeline, but some platform-specific technologies that have their counterparts in different provider offerings are used.

Microsoft Azure is chosen as an experiment since Profinit EU runs most of its infrastructure on Amazon Web Services (AWS). This choice allows them to evaluate the viability of the Microsoft Azure cloud computing provider better and avoids running everything under one provider.

[7] Stemming from the tech industry slang term, where the internet was represented as a cloud in the diagrams [67].

5.1.2 Kubernetes

Based on the requirements of the tools used in the pipeline and the need for vendor independence and scalability, Kubernetes has been chosen as the system for pipeline and application deployment.

Kubernetes is the most widely used orchestration platform for managing containerized applications [69]. Initially developed by Google, it is currently an open-source project under the Cloud Native Computing Foundation (CNCF). It provides automation when it comes to deploying, managing, and scaling containerized applications.

Containers provide an isolated environment to run applications where the code is packaged with all its dependencies to enable portability between different computing environments. Kubernetes utilizes these containers using different container runtimes to run a set of containers in Pods. A Pod is the smallest deployment unit in the Kubernetes system.

Kubernetes contains various objects in its API, each with its purpose. These objects hold information about the expected and current state of the objects' operation.

The deployment of a Kubernetes system also called a Kubernetes cluster, is often distributed on multiple nodes. In each deployment, there needs to be a Control Plane component, managing the nodes, exposing the Kubernetes API, and overall ensuring smooth operation of the cluster. One significant advantage of Kubernetes is that it can be run on various infrastructure setups. This advantage makes the technology a great choice since it does not lock us into just one infrastructure vendor.

Deployments of applications or workloads to Kubernetes are defined using declarative manifests where the objects the developer wants to have deployed are specified. These manifests can be JSON or YAML files following a defined schema.

Interaction with the Kubernetes cluster is done through the API exposed by the cluster's Control Plane. The developers often use the command line tool kubectl for the interaction. This tool abstracts the API into command options. [70]

This technology is very complex and has many intricacies, so we avoid going deeper, given the limited scope of the thesis.

5.2 Tools

Multiple tools are utilized to implement the pipeline and migrate the application. Here is a short overview of each of the tools and their purpose.

5.2.1 Terraform

To allow repeatable builds and safe changes to the infrastructure, the Infrastructure as Code (IaC) tool Terraform is used. Terraform allows us to efficiently manage the resources used in the cloud and deployed on the provisioned Kubernetes cluster.

The infrastructure is defined using Terraform configuration files. These configuration files use declarative Terraform language to describe the intended goal. Using the Terraform configuration files, Terraform allows us to plan and make changes to the infrastructure. After making the changes, Terraform stores the state of the provisioned resources. The state is then taken into account when planning new changes to the infrastructure. [71]

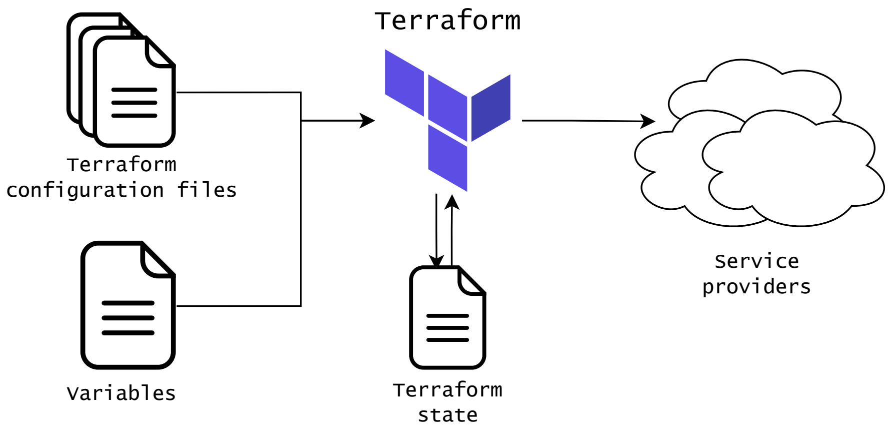


Figure 5.1: Terraform takes configuration files with variables and, with the information stored in the state, it plans and provisions or changes resources provided by service providers.

Terraform creates and manages resources in services using APIs abstracted by Terraform providers. These providers allow us to create necessary resources in the Microsoft Azure cloud, on the Kubernetes Cluster provisioned in the cloud, and in Profinit's GitLab instance.

In the implementation, the resources are split into Terraform modules, which enable the encapsulation of the infrastructure parts. The modules can be deployed separately if their dependencies are satisfied.

5.2.2 Helm

Helm, as a Terraform provider, is utilized to deploy tools for the pipeline and application where possible.

Helm enables simple installation, upgrading, and application management for Kubernetes applications. Applications packaged using Helm are called charts. These Helm charts contain all Kubernetes object definitions necessary to run the application.

Kubernetes object definitions in the Helm charts are often templated to allow configuration and portability of application deployment. This way, applications packaged using Helm can be used in many different environments; the only difference is the configuration values provided. [72]

Many of the tools chosen for the pipeline provide either official Helm charts or third-party Helm charts developed and maintained by the community or other companies. Using these charts with the Terraform provider allows us to easily deploy the tools to the provisioned Kubernetes cluster.

Some of the tools in the pipeline do not provide Helm charts. A custom definition of the Kubernetes objects necessary to run the application is written for these tools. Creating charts for these applications is not viable since a lot of experience and time is necessary to create a maintainable and configurable chart.

5.2.3 Kustomize

Kustomize is selected as the tool to deploy the provided application to the Kubernetes cluster. It allows the developer to parameterize vanilla Kubernetes manifests without creating complex templates of the object definitions, as is the case with Helm. Using Kubernetes manifests with Kustomize is more straightforward, but on the other hand, it does not provide such configuration freedom and portability as Helm does.

The great thing about Kustomize is that its features are also built into kubectl, the default command line utility to communicate with a Kubernetes cluster. [73]

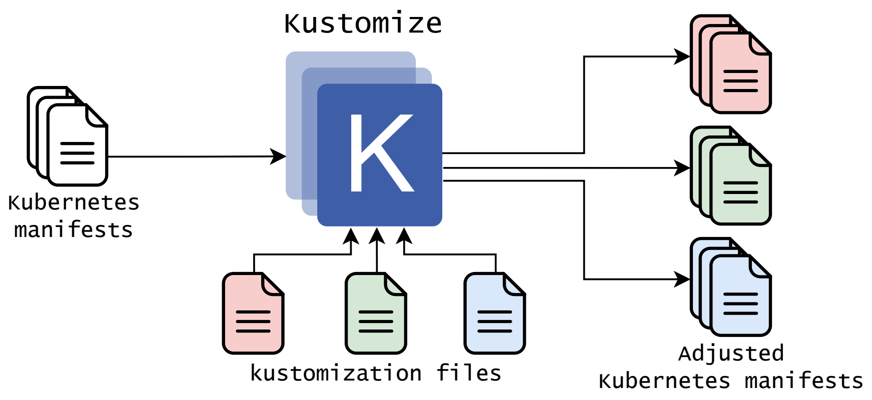


Figure 5.2: Based on the configuration specified in kustomization files, vanilla Kubernetes manifests are processed by Kustomize and adjusted.

In the deployment of the application, Kustomize is used as a simple way to enable the application to be deployed to multiple environments. These environments can include, for example, a development or production environment. Using Kustomize removes the need for duplication, allowing us to write the manifests only once and adjust them per environment.

5.2.4 GitLab CI/CD

As previously mentioned in section 1.4 describing the principles of MLOps, one principle is CI/CD automation. In the overview of the pipeline design, the CI/CD tool is left out of the diagram. The reason for this decision is that the tool choice is highly dependent on what the company is using, and it is not deterministic to the overall pipeline function. In this case, to integrate all of the services and enable the automation of CI and CD, GitLab CI/CD is used.

GitLab CI/CD is a CI/CD tool that allows us to write CI/CD pipelines consisting of jobs automating the tasks that need to be done to build, test, and deploy the application and various other tasks. Steps in the jobs are run on a GitLab runner, which provides an environment in which the commands defined in a job are executed. [74]

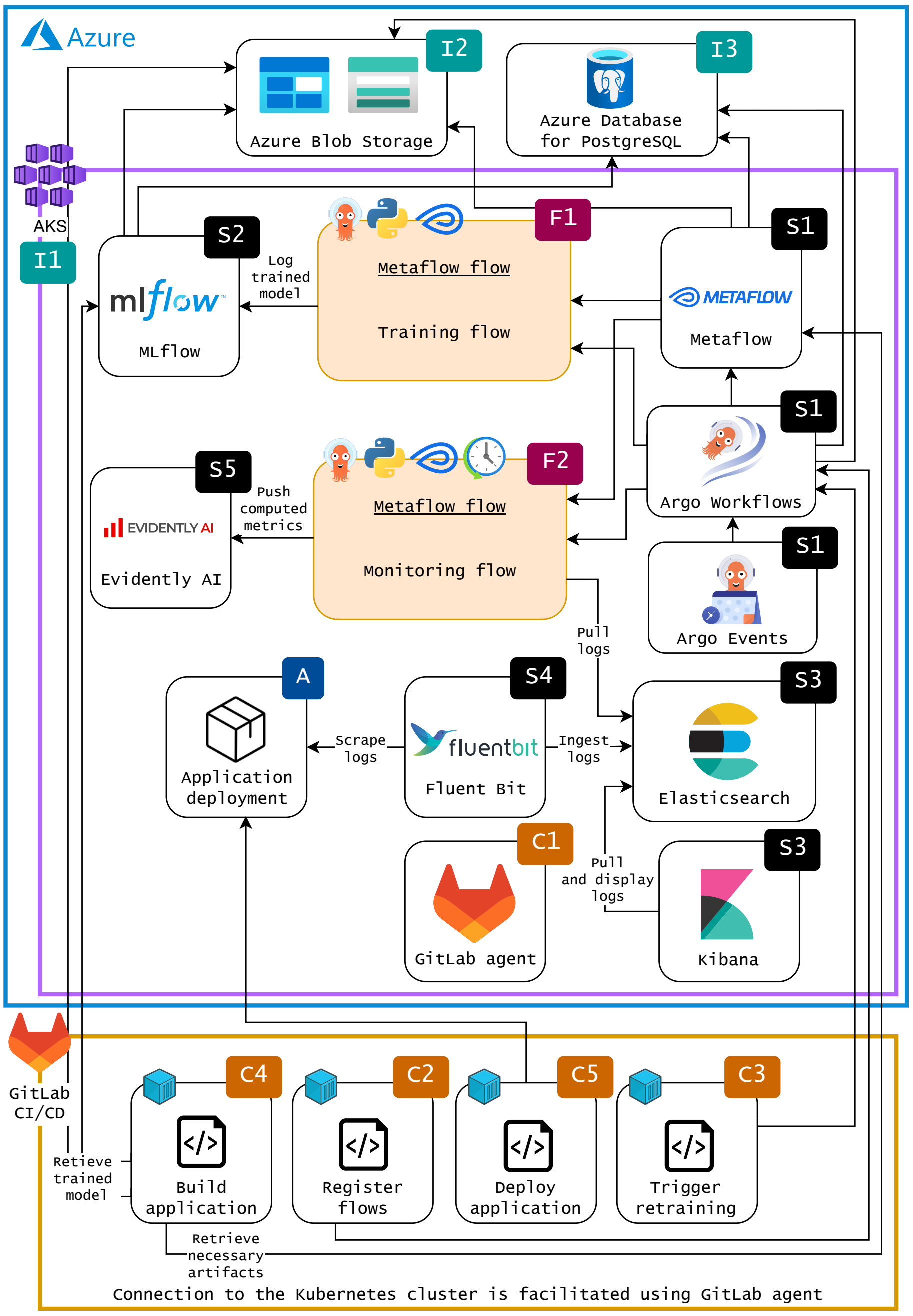


Figure 5.3: Diagram of the implemented pipeline. Application deployment and certain deployment details are omitted due to the complexity of the solution.

5.3 Pipeline Implementation

In the diagram 5.3, there is an overview of the deployed pipeline. Resources in the diagram are labeled and split into different groups to form logical units.

The beginning letters of the label concern the group, and the number is used to order the resources.

Resources for the pipeline designed in chapter 4 are defined and deployed using Terraform. The pipeline implementation is split into two Terraform modules, the `infra` module **(I)** and `services` module **(S)**. Implementation is split to allow for portability. With few adjustments, the `services` module would be deployable on top of different infrastructure resources.

Flows of the Metaflow workflow orchestrator are Python scripts where a workflow is defined. These flows **(F)** play a significant role in pipeline operation, so they are depicted in the diagram as well.

Since the provided application needs to be migrated to the cloud, there will be a section devoted to the application migration and deployment itself. In this diagram, the application **(A)** is abstracted to show the interactions in the pipeline.

CI/CD component **(C)** is also illustrated in the diagram. This component enables the CI/CD automation specific to our case. In this case, the CI/CD is covered using GitLab CI/CD. An overview of CI/CD automation will be provided in a future section.

5.3.1 infra Terraform Module

The `infra` Terraform module **(I)** contains the definitions of underlying resources that the MLOps tools are dependent on. These resources are all deployed in the Microsoft Azure cloud.

- **(I1) Azure Kubernetes Service (AKS).** Since Kubernetes is selected as an underlying system for the pipeline and application deployment, AKS is used to provision a Kubernetes cluster. An advantage of using AKS is the reduced complexity of Kubernetes management since many of the management tasks are offloaded to Azure. The cluster utilizes only one worker node and cheap spot nodes to minimize the costs during this experiment.

- **(I2) Azure Blob Storage.** Metaflow, MLflow, and Argo need object storage to save the artifacts. Azure Blob Storage in the Azure Storage Account is used as the object storage for these services.

- **(I3) Azure Database for PostgreSQL.** Multiple of the services used in the MLOps pipeline require a PostgreSQL database. The managed Azure Database for PostgreSQL – Flexible Server is deployed to avoid the management complexity of running a database in the Kubernetes cluster and to avoid deployment difficulties. Multiple databases have to be created inside the managed PostgreSQL resource. One database is for Metaflow, another for Argo, and the last for MLflow.

5.3.2 services Terraform Module

Services for the MLOps pipeline implementation are defined in the `services` Terraform module **(S)**. These services are described in the chapter 4 concerning the pipeline design. The `infra` Terraform module needs to be deployed before this module. Terraform automatically figures out this dependence.

- **(S1) Metaflow, Argo Workflows, Argo Events.** The main component of the pipeline is the workflow orchestrator. Metaflow, in combination with Argo Workflows and Argo Events, provides this functionality. Argo Workflows and Argo Events provide Helm charts for deploying these services. These Helm charts were deployed using Terraform Helm provider. Metaflow was deployed based on Kubernetes manifest definitions in Terraform [75] provided by Outerbounds – the maintainers of Metaflow.

  Metaflow and Argo Workflows store data in the provisioned PostgreSQL database and Azure Blob Storage object storage.

- **(S2) MLflow.** Experiment tracking tool MLflow is deployed using a Helm chart with the Terraform Helm provider. MLflow connects to the provisioned PostgreSQL database and stores data in the Azure Blob Storage object storage.

- **(S3) Elasticsearch, Kibana.** These monitoring tools are deployed using the official Kubernetes controller – Elastic Cloud on Kubernetes. This controller is first deployed using a Helm chart, and then another Helm chart deploys the Elasticsearch and Kibana instances themselves. In the configuration, the Elasticsearch is configured to run with minimal resource requirements to save costs during this experiment. Elasticsearch persists its data using a PersistentVolumeClaim Kubernetes object.

- **(S4) Fluent Bit.** Similarly to other tools, the log aggregation and forwarding tool Fluent Bit is also deployed using a Helm chart with the Terraform Helm provider. Fluent Bit is automatically configured to retrieve logs from the Pods running in the cluster and forward them to the Elasticsearch service.

- **(S5) Evidently AI.** The model monitoring tool, Evidently AI, is deployed using custom Kubernetes object definitions for Terraform Kubernetes provider. These definitions contain the Deployment of the application, a Service to expose it, and a PersistentVolumeClaim to persist its data.

5.3.3 Metaflow Flows

In the diagram 5.3 of the implemented pipeline, there are two Metaflow flows depicted **(F1, F2)**. The amount of registered flows and their purpose can vary depending on the current needs of the developers and the application. These two are displayed in the diagram because flows with similar purposes are going to be present in every pipeline implementation, and they help illustrate the workflow of the pipeline.

Metaflow flows are Python scripts that utilize the API provided by Metaflow to enable workflow orchestration. These flows can be registered automatically to the Kubernetes cluster in a CI/CD pipeline. There is also an option to register flows manually as the developer experiments with their implementation.

In the implementation of the pipeline, the suggested flows should have the following purposes:

- **(F1) Training Flow.** As the name suggests, this Metaflow flow covers the training of the ML model. This means the flow loads the data necessary for the training and starts the experiment training run. Data for the training run can be versioned using Metaflow's built-in versioning capabilities or pushed into some object storage. During the training run, metadata, metrics, and other artifacts should be stored in the MLflow experiment. Finally, when the model training is done, the resulting model should be pushed into the MLflow's model repository.

- **(F2) Monitoring Flow.** This flow enables monitoring of the model's performance. Model inference logs are scraped and ingested into Elasticsearch using Fluent Bit. This flow then, on a set schedule or with another trigger, pulls the stored data from the Elasticsearch service, computes the monitoring metrics, and pushes them into Evidently AI monitoring UI. Alerts based on the derived information can also be sent to developers from this flow.

5.4 Application Migration

One of the requirements set by Profinit EU is that the application described in the chapter 3 is supposed to run in the cloud. Since a Kubernetes cluster **(I1)** is already provisioned for the described MLOps pipeline for the application, we agreed to deploy the application to the same Kubernetes cluster.

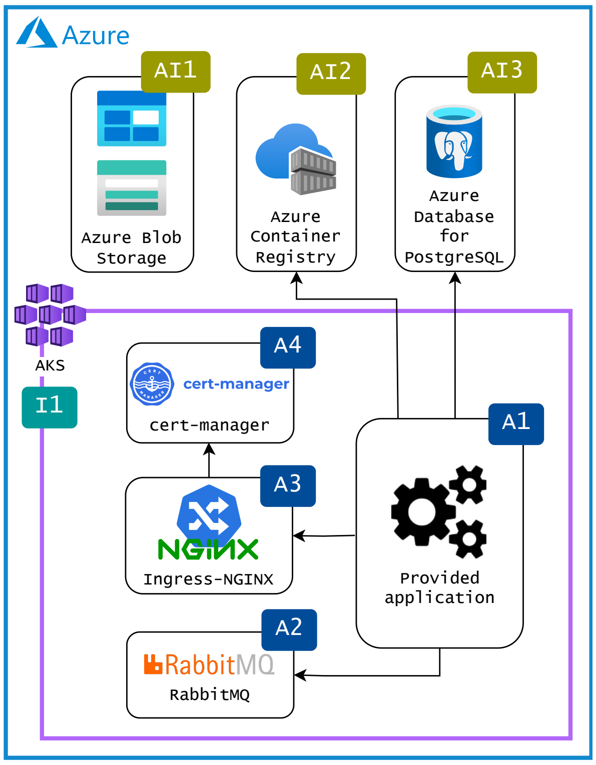


Figure 5.4: Diagram of the resources for the migrated application.

The application migration is split into two parts when moving into the cloud environment. The first part is the resources necessary for the application to run. The second part is the application deployment itself.

Resources necessary for the application migration are created using Terraform. To separate the resources for the application from the ones for the pipeline, they are extracted into the `application` Terraform module.

The application is continuously deployed using GitLab CI/CD pipeline by applying Kubernetes manifests configured per environment using Kustomize.

5.4.1 application Terraform Module

We provision the necessary resources in the cloud and inside the Kubernetes cluster using Terraform. All of the resources for the application (except the Kubernetes cluster itself **(I1)**, provisioned in the pipeline `infra` module) are provisioned inside the `application` module in the Terraform project.

As described in the application architecture overview, its components include the Flask web application, Celery worker, RabbitMQ message broker, and PostgreSQL database. We provision some of these components and other resources to allow the application's deployment inside the Kubernetes cluster. Here is an overview of the resources provisioned in the `application` Terraform module.

- **(AI1) Azure Blob Storage.** Azure Blob Storage object storage available in the Azure Storage Account stores the data necessary for the training and static asset generation when building the application. This resource helps to avoid storing the data in the Git repository.

- **(AI2) Azure Container Registry.** Built images of the application need to be pushed to a container registry to be pulled into the Kubernetes cluster and deployed. Azure Container Registry is provisioned exactly to solve this problem. In practice, this resource allows the storage of Docker and Open Container Initiative (OCI) images and all other OCI artifacts.

- **(AI3) Azure Database for PostgreSQL.** Similarly to the database service used for the services database, Azure Database for PostgreSQL is provisioned. Connection to this database is publicly available since some of the management tasks for the application need to be done manually.

- **(A2) RabbitMQ.** This resource allows the Flask web application and the Celery worker to communicate. RabbitMQ's proper deployment and deployment management are complex tasks, but choosing a Helm chart to deploy this resource simplifies these tasks. This resource is deployed into the Kubernetes cluster using a Helm chart and the Terraform Helm provider.

- **(A3) Ingress-NGINX.** An Ingress Kubernetes resource exposes the application from inside the cluster. Ingress-NGINX is chosen as the Ingress Controller due to its popularity and portability between cloud environments [76]. This resource is deployed into the Kubernetes cluster using a Helm chart and the Terraform Helm provider.

- **(A4) cert-manager.** The cert-manager is deployed in the Kubernetes cluster to allow for secure communication between the users and the application. cert-manager obtains TLS certificates from various issuers [77]; in our case, it is the Let's Encrypt issuer. These certificates can then be used with the Ingress resource. This resource is deployed into the Kubernetes cluster using a Helm chart and the Terraform Helm provider.

5.4.2 Application Deployment

Vanilla Kubernetes manifests are written to deploy the application itself **(A1)** into the Kubernetes cluster, and the Kustomize overlays are used to adjust these manifests per environment. Since the application already uses Docker for containerization, the existing Docker images can be used in the Kubernetes cluster.

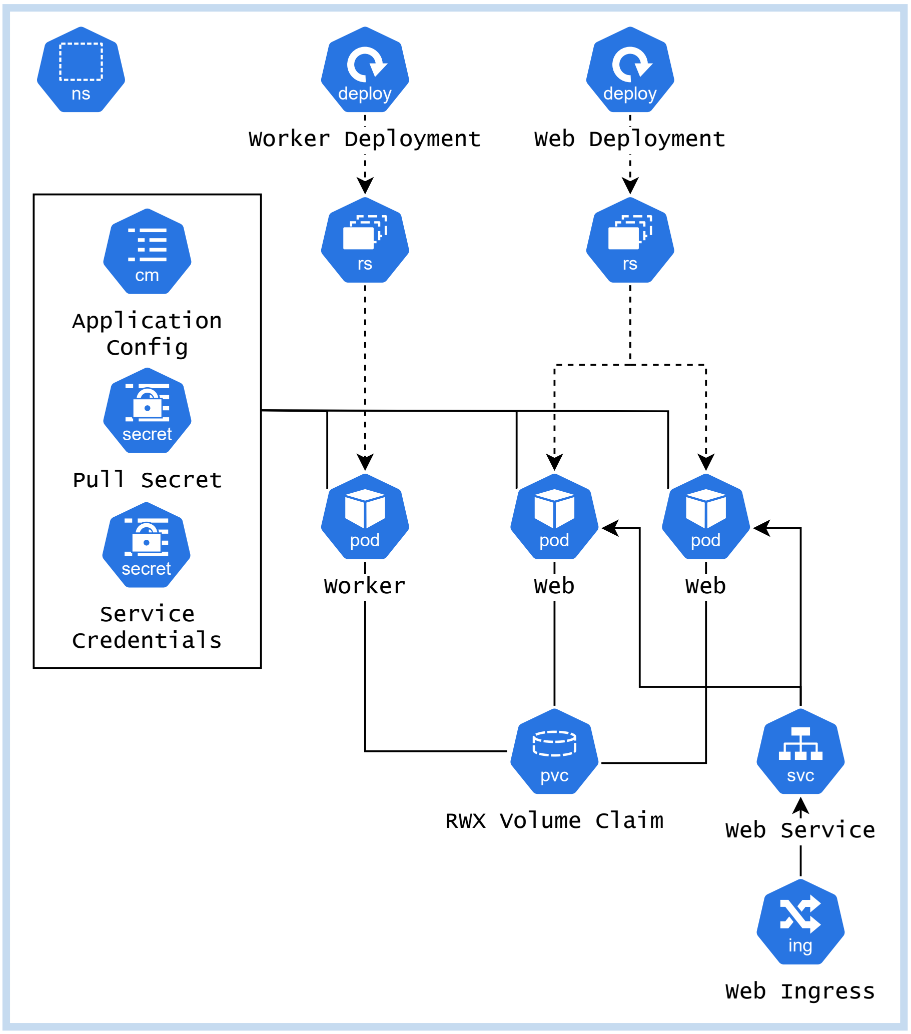


Figure 5.5: Diagram of the provided application deployed in Kubernetes cluster

The application consists of two Deployments managing ReplicaSets. One of the Deployments is for the Flask web application, and the other is for the Celery worker. Configuration for the application is provided using ConfigMaps.

Credentials to connect to the database and message broker and pull secret for the container registry are provided using Secrets. Each Flask web application pod and the Celery worker pods have a ReadWriteMany volume mounted to provide the shared folder. This volume is claimed by a PersistentVolumeClaim, which is provisioned on Azure using one of the provided StorageClasses. Lastly, the Flask web application is exposed using a Service in the cluster, and Ingress provides the external access.

Docker images for the deployment of the pods are pulled from the Azure Container Registry. The application connects to the RabbitMQ running in the cluster and the managed PostgreSQL database.

This deployment is depicted in the diagram 5.5. Connection to the external services is not shown in the diagram.

5.5 CI/CD Automation

One of the principles of MLOps described in the section 1.4 is CI/CD automation. The CI/CD automation component **(C)** in this implementation is GitLab CI/CD, as mentioned previously.

GitLab CI/CD runs its jobs on a runner provided by Profinit EU. This runner communicates with the Kubernetes cluster using a GitLab agent **(C1)**, which is deployed using an official Helm chart with Terraform Helm provider. Terraform also creates necessary CI/CD environment variables for the jobs.

The diagram 5.3 highlights only the most important jobs. These jobs can be used as suggestions when implementing pipelines on different projects. Various other jobs can be added based on the requirements. Here is a short overview of each of the jobs:

- **(C2) Register flows.** When a developer changes the definition of any Metaflow flows, a CI/CD job automatically registers the new version in the Kubernetes cluster.

- **(C3) Trigger retraining.** Retraining can be triggered based on multiple criteria, as described in the previous chapters. One of the ways to trigger the retraining can be using a CI/CD job, which runs, for example, automatically on changes or when a developer triggers it from GitLab's UI.

- **(C4) Build application.** A build job for the final application can be present in the implemented pipeline. This job can, for example, use the MLflow's model registry to download a tagged model or retrieve necessary artifacts from the Metaflow flow's run. Details of this job are left out because they are specific to the provided application.

- **(C5) Deploy application.** Finally, a job to deploy the application to the cluster should be present. In this case, this job applies the Kubernetes manifests adjusted per environment using Kustomize.

5.6 Pipeline Summary

To summarize the pipeline implementation and give an idea of the process when using the pipeline, below is an ordered list of steps happening in the pipeline.

1. Metaflow flows are registered into the Kubernetes cluster.
2. Training flow is triggered. The model is trained and saved with its artifacts to MLflow.
3. Model training results can be manually analyzed in the MLflow UI, and if the newly trained model is better than the older ones, it can be selected for deployment.
4. Application is built. During the build, the model chosen for deployment is downloaded with its artifacts.
5. The built application is deployed to the Kubernetes cluster.
6. Application runs, and the logs from its usage are scraped using Fluent Bit and ingested into Elasticsearch.
7. Monitoring flow pulls the saved logs, calculates insights using the Evidently AI library, and pushes the derived information into the Evidently AI monitoring UI. This flow can also send alerts to developers based on the derived information.
8. The Developer can then act based on alerts received or the information in the Evidently AI monitoring UI.

Steps happening in the pipeline can be adjusted based on the needs of specific projects thanks to the pipeline's flexibility. The described steps are just one way to implement the pipeline used in the prototype for the provided application.

Chapter 6

Discussion

The last few chapters provided an overview of MLOps pipeline design and its implementation. The designed pipeline was successfully deployed and used with the application provided by Profinit EU. This process has provided us with valuable insights, which are written in the following sections.

6.1 Comparison with Original State

Compared to the original state of the application, many of the associated manual activities with the deployment are now automated. The application is deployed continuously on changes. The model for the application can be easily retrained on new data, and all trained models are versioned. Metrics are collected from application usage to assess performance regularly. On the other hand, the deployment is much more complex, with many tools involved. It is also an expensive solution due to the amount of resources used in the cloud environment.

6.2 Downsides

There are multiple downsides related to using the designed pipeline. Some of them were already mentioned when describing the pipeline design. Below is a comprehensive list of the downsides with short elaborations.

- Many of the tools in the pipeline do not provide user isolation. When a user has access to the tool, they have access to everything that the certain tool manages. This way, there could be unwanted consequences for mistakes, or a malicious actor could take advantage of this fact.
- Due to the amount of tools, the pipeline is overly complex. Because of this, it may be difficult for developers to use this pipeline. Usage requires knowledge of multiple tools and their interactions, which many developers do not have.
- For small projects, using this pipeline may be costly. Another thing that does not help is that valuable resources are used for the Kubernetes cluster management. Consider using this pipeline when there is a need for scalability and resource-intensive workflows.

- There is a need for a custom authentication layer. For reference, in the implementation for Profinit EU, authentication is solved using OAuth2 Proxy with the Ingress-NGINX external OAuth authentication access method. Their GitLab instance is used as an OAuth 2.0 authentication identity provider for the OAuth2 Proxy. Kubectl's port forwarding functionality is used for programmatic access to the APIs, which is not an elegant solution.

6.3 Future Work

This pipeline design and implementation should mainly demonstrate how to apply the principles of MLOps. Of course, it is not directly applicable to all existing cases and needs to be adapted to the specific requirements of a given case. Here, we see some possible work that could be done in the future to improve the solution and provide a better developer experience. This work is described in the following points.

- The source code provided in the archive of this thesis focuses on a specific implementation for the provided application. On the basis of the design and this source code, it would be useful to create a template on which implementations for other projects could be based.
- As mentioned in the downsides, the authentication is currently not solved elegantly. A way to authenticate in the pipeline and provide both browser access to the UIs and programmatic access to the APIs would be a great addition to this pipeline.
- For projects where the pipeline is not being retrofitted, it would be a good idea to enhance the design with recipes for model deployment options. This way, a recipe can be chosen and used in the pipeline without much hassle when a new solution is developed.
- Monitoring resource usage would be a beneficial addition to this pipeline. There are multiple off-the-shelf solutions for monitoring inside a Kubernetes cluster. Implementing any of these solutions should be considered in the future.

Conclusion

The main goal of the thesis was to design and implement an MLOps pipeline. Implementing the MLOps pipeline meant deploying the designed pipeline in practice and applying it to an application provided by Profinit EU. Before the pipeline could be applied to the application, the application had to be migrated to a cloud environment. Other goals that supported the primary goal were to get a good grasp of the MLOps paradigm and its principles and to get an overview of tools available in the MLOps landscape.

In order to achieve these goals, the theoretical concepts of ML and MLOps were first introduced. Following the introduction, there was an overview of open-source tools and their purposes in the MLOps paradigm. The application provided by Profinit EU was then analyzed. After this analysis, it was concluded that the application does not necessitate an extensive solution. However, it was chosen to proceed with the pipeline design to gain knowledge. An extensible and portable pipeline was designed using open-source tools. Finally, the designed pipeline was implemented and successfully used with the application, which was migrated to the cloud environment.

The work on this thesis proved to be very difficult. This difficulty stems mainly from the sheer amount of new tools that we had to familiarize ourselves with and use. Another thing is that when working with infrastructure, the feedback loops on changes are long.

In conclusion, this thesis successfully designed and implemented a pipeline to apply the MLOps principles to the application provided by Profinit EU. The pipeline design provides a starting point for anyone wanting to enhance the development of applications utilizing ML by applying MLOps principles.

Bibliography

1. SHAHI, Bhanu. *Introduction To Machine Learning* [online]. 2021-06. [visited on 2024-03-19]. Available from: `https://bhanushahi.medium.com/introduction-to-machine-learning-900404cc8f3e`.
2. CARBTREE, Matt. *What is Machine Learning? Definition, Types, Tools & More* [online]. 2023-07. [visited on 2024-03-20]. Available from: `https://www.datacamp.com/blog/what-is-machine-learning`.
3. BUSS, Sebastian; NÖLDEKE, Geeske; BECKER, Dennis; BLUMTRITT, Christoph; DANIELS, Marcos; STRIAPUNINA, Ksenia. *Statista Digital Economy Compass 2019* [online]. 2019-04. [visited on 2024-03-17]. Available from: `https://cdn.statcdn.com/download/pdf/DigitalEconomyCompass2019.pdf`.
4. HUYEN, Chip. *Designing Machine Learning Systems*. USA: O'Reilly Media, 2022. ISBN 978-1801819312.
5. FOWLER, Martin. *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley, 1999. ISBN 0-201-48567-2.
6. SCULLEY, D; HOLT, Gary; GOLOVIN, Daniel; DAVYDOV, Eugene; PHILLIPS, Todd; EBNER, Dietmar; CHAUDHARY, Vinay; YOUNG, Michael; DENNISON, Dan. Hidden Technical Debt in Machine Learning Systems. *NIPS*. 2015, pp. 2494–2502.
7. KARPATHY, Andrej. *Building the Software 2.0 Stack* [online]. Youtube, 2018-08. [visited on 2024-03-17]. Available from: `https://youtu.be/y57wwucbXR8?si=ZJTs8IgG0W97vl9A&t=522`.
8. DR. VISENGERIYEVA, Larysa; KAMMER, Anja; BÄR, Isabel; KNIESZ, Alexander; PLÖD, Michael. *Why you Might Want to use Machine Learning* [online]. 2019. [visited on 2024-03-18]. Available from: `https://ml-ops.org/content/motivation`.
9. TREVEIL, M.; OMONT, N.; STENAC, C.; LEFEVRE, K.; PHAN, D.; ZENTICI, J.; LAVOILLOTTE, A.; HEIDMANN, L.; MIYAZAKI, M. *Introducing MLOps: How to Scale Machine Learning in the Enterprise*. O'Reilly Media, Incorporated, 2020. ISBN 9781492083290. Available also from: `https://learning.oreilly.com/library/view/introducing-mlops/9781492083283/`.

10. STAFF, VB. *Why do 87% of data science projects never make it into production?* [online]. 2019-07. [visited on 2024-03-17]. Available from: `https://venturebeat.com/ai/why-do-87-of-data-science-projects-never-make-it-into-production/`.

11. DR. VISENGERIYEVA, Larysa; KAMMER, Anja; BÄR, Isabel; KNIESZ, Alexander; PLÖD, Michael. *End-to-end Machine Learning Workflow* [online]. 2021. [visited on 2024-03-27]. Available from: `https://ml-ops.org/content/end-to-end-ml-workflow`.

12. DR. VISENGERIYEVA, Larysa; KAMMER, Anja; BÄR, Isabel; KNIESZ, Alexander; PLÖD, Michael. *Three Levels of ML Software* [online]. 2021. [visited on 2024-04-09]. Available from: `https://ml-ops.org/content/three-levels-of-ml-software`.

13. MORGUNOV, Anton. *The Life Cycle of a Machine Learning Project: What Are the Stages?* [online]. 2023-08. [visited on 2024-04-09]. Available from: `https://neptune.ai/blog/life-cycle-of-a-machine-learning-project`.

14. ACHARYA, Akruti. *Training, Validation, Test Split for Machine Learning Datasets* [online]. 2023-06. [visited on 2024-03-28]. Available from: `https://encord.com/blog/train-val-test-split/`.

15. AMAZON WEB SERVICES. *Machine Learning Lens: Model training and tuning* [online]. 2023-07. [visited on 2024-03-28]. Available from: `https://docs.aws.amazon.com/wellarchitected/latest/machine-learning-lens/model-training-and-tuning.html`.

16. DOWLING, Jim. *Guide to File Formats for Machine Learning: Columnar, Training, Inferencing, and the Feature Store | by Jim Dowling | Towards Data Science* [online]. 2019-10. [visited on 2024-03-28]. Available from: `https://towardsdatascience.com/guide-to-file-formats-for-machine-learning-columnar-training-inferencing-and-the-feature-store-2e0c3d18d4f9`.

17. EVIDENTLY AI TEAM. *Model monitoring for ML in production: a comprehensive guide* [online]. [visited on 2024-04-09]. Available from: `https://www.evidentlyai.com/ml-in-production/model-monitoring`.

18. AMERSHI, Saleema; BEGEL, Andrew; BIRD, Christian; DELINE, Robert; GALL, Harald; KAMAR, Ece; NAGAPPAN, Nachiappan; NUSHI, Besmira; ZIMMERMANN, Thomas. Software Engineering for Machine Learning: A Case Study. In: 2019, pp. 291–300. Available from DOI: `10.1109/ICSE-SEIP.2019.00042`.

19. STUDER, Stefan; BUI, Binh; DRESCHER, Christian; HANUSCHKIN, Alexander; WINKLER, Ludwig; PETERS, Steven; MÜLLER, Klaus-Robert. Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology. *Machine Learning and Knowledge Extraction*. 2021, vol. 3, pp. 392–413. Available from DOI: `10.3390/make3020020`.

20. WIRTH, Rüdiger; HIPP, Jochen. CRISP-DM: Towards a Standard Process Model for Data Mining. In: 2000. Available also from: `https://api.semanticscholar.org/CorpusID:1211505`.

21. TESTI, Matteo; BALLABIO, Matteo; FRONTONI, Emanuele; IANNELLO, Giulio; MOCCIA, Sara; SODA, Paolo; VESSIO, Gennaro. MLOps: A Taxonomy and a Methodology. *IEEE Access*. 2022, vol. 10, pp. 63606–63618. Available from DOI: 10.1109/ACCESS.2022.3181730.

22. KREUZBERGER, Dominik; KÜHL, Niklas; HIRSCHL, Sebastian. Machine Learning Operations (MLOps): Overview, Definition, and Architecture. *IEEE Access*. 2023, vol. 11, pp. 31866–31879. Available from DOI: 10.1109/ACCESS.2023.3262138.

23. CDF SPECIAL INTEREST GROUP - MLOPS. *MLOps Roadmap 2022* [online]. 2022-11. [visited on 2024-03-20]. Available from: https://github.com/cdfoundation/sig-mlops/blob/2d9284821a2ae91b2118c0e6c7de5a3d2fb769eb/roadmap/2022/MLOpsRoadmap2022.md.

24. GOOGLE. *MLOps – Explore – Google Trends* [online]. [visited on 2024-03-20]. Available from: https://trends.google.com/trends/explore?date=all&q=%2Fg%2F11h1vbjpbg&hl=en.

25. FAUBEL, Leonhard; SCHMID, Klaus; EICHELBERGER, Holger. MLOps Challenges in Industry 4.0. *SN Computer Science*. 2023, pp. 1–11. Available from DOI: 10.1007/s42979-023-02282-2.

26. GITLAB. *What is DevOps?* [online]. 2023-01. [visited on 2024-03-29]. Available from: https://about.gitlab.com/topics/devops/.

27. DREMIO TEAM. *Introduction to Data Engineering* [online]. [visited on 2024-03-29]. Available from: https://www.dremio.com/resources/guides/intro-data-engineering/.

28. SHALEV-SHWARTZ, Shai. *Online Learning and Online Convex Optimization*. 2012. Available from DOI: 10.1561/2200000018.

29. GOOGLE CLOUD. *MLOps: Continuous delivery and automation pipelines in machine learning* [online]. 2023-05. [visited on 2024-03-20]. Available from: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning.

30. CLOUDFLARE, INC. *What is vendor lock-in? | Vendor lock-in and cloud computing* [online]. [visited on 2024-04-02]. Available from: https://www.cloudflare.com/learning/cloud/what-is-vendor-lock-in/.

31. OLADELE, Stephen. *MLOps Landscape in 2024: Top Tools and Platforms* [online]. 2024-03. [visited on 2024-04-02]. Available from: https://neptune.ai/blog/mlops-tools-platforms-landscape.

32. DUTA, Grig. *Top End-to-End MLOps Platforms and Tools in 2024* [online]. 2024-02. [visited on 2024-04-02]. Available from: https://www.qwak.com/post/top-mlops-end-to-end.

33. THE KUBEFLOW AUTHORS. *Documentation* [online]. [visited on 2024-04-03]. Available from: https://www.kubeflow.org/docs/.

34. LEWINSON, Eryk. *Best ML Workflow and Pipeline Orchestration Tools 2024* [online]. 2024-04. [visited on 2024-04-11]. Available from: https://dagshub.com/blog/best-machine-learning-workflow-and-pipeline-orchestration-tools/.

35. FLYTE AUTHORS. *Flyte Documentation* [online]. 2024-03. [visited on 2024-04-12]. Available from: https://docs.flyte.org/en/v1.11.0/.

36. OUTERBOUNDS; METAFLOW CONTRIBUTORS. *Metaflow Docs* [online]. [visited on 2024-04-11]. Available from: `https://docs.metaflow.org/`.

37. NETFLIX TECHNOLOGY BLOG. *Open-Sourcing a Monitoring GUI for Metaflow, Netflix's ML Platform* [online]. 2021-10. [visited on 2024-04-11]. Available from: `https://netflixtechblog.com/open-sourcing-a-monitoring-gui-for-metaflow-75ff465f0d60`.

38. ZENML. *ZenML Documentation* [online]. 2024-04. [visited on 2024-04-11]. Available from: `https://docs.zenml.io/v/0.56.3`.

39. DVC.AI. *DVC Documentation* [online]. [visited on 2024-04-13]. Available from: `https://dvc.org/doc`.

40. PREFECT TECHNOLOGIES, INC. *Perfect Documentation* [online]. 2024-04. [visited on 2024-04-13]. Available from: `https://docs.prefect.io/2.17.0/`.

41. ARGO PROJECT AUTHORS. *Argo Workflows Documentation* [online]. 2024-04. [visited on 2024-04-13]. Available from: `https://argo-workflows.readthedocs.io/en/release-3.5/`.

42. THE APACHE SOFTWARE FOUNDATION. *Apache Airflow Documentation* [online]. [visited on 2024-04-13]. Available from: `https://airflow.apache.org/docs/`.

43. MOHANDAS, Goku. Tracking – Made With ML. 2023. Available also from: `https://madewithml.com/courses/mlops/experiment-tracking/`.

44. KLUGE, Kilian; JENKNER, Patrycja. *Best Tools for ML Experiment Tracking and Management in 2024* [online]. 2024-04. [visited on 2024-04-12]. Available from: `https://neptune.ai/blog/best-ml-experiment-tracking-tools`.

45. SOGOMONIAN, Gev; ARAKELYAN, Gor, et al. *Aim v3.19.2 Documentation* [online]. 2024-03. [visited on 2024-04-13]. Available from: `https://aimstack.readthedocs.io/en/latest/index.html`.

46. MLFLOW PROJECT AUTHORS. *MLflow 2.11.3 Documentation* [online]. 2024-03. [visited on 2024-04-13]. Available from: `https://mlflow.org/docs/latest/index.html`.

47. SACRED CONTRIBUTORS. *Sacred 0.8.5 Documentation* [online]. 2023-11. [visited on 2024-04-13]. Available from: `https://sacred.readthedocs.io/en/stable/index.html`.

48. GIT LFS CONTRIBUTORS. *Git Large File Storage* [online]. 2024-03. [visited on 2024-04-13]. Available from: `https://git-lfs.com/`.

49. PACHYDERM. *Pachyderm Documentation* [online]. 2024. [visited on 2024-04-13]. Available from: `https://docs.pachyderm.com/`.

50. PIENAAR, Willem; BALSO, Mike Del. *What is a Feature Store?* [online]. 2021-01. [visited on 2024-04-13]. Available from: `https://feastsite.wpenginepowered.com/blog/what-is-a-feature-store/`.

51. FEAST AUTHORS. *Feast Documentation* [online]. 2023-04. [visited on 2024-04-13]. Available from: `https://docs.feast.dev/v/v0.36-branch`.

52. HOPSWORKS CONTRIBUTORS. *Hopsworks Documentation* [online]. 2024-03. [visited on 2024-04-13]. Available from: `https://docs.hopsworks.ai/3.7/`.

53. AKINREMI, Bunmi. *Best Tools for Model Tuning and Hyperparameter Optimization* [online]. 2023-11. [visited on 2024-04-13]. Available from: `https://neptune.ai/blog/best-tools-for-model-tuning-and-hyperparameter-optimization`.

54. OPTUNA CONTRIBUTORS. *Optuna 3.6.1 Documentation* [online]. 2024-04. [visited on 2024-04-13]. Available from: `https://optuna.readthedocs.io/en/v3.6.1/`.

55. THE RAY TEAM. *Ray 2.9.3 Documentation* [online]. 2024-02. [visited on 2024-04-13]. Available from: `https://docs.ray.io/en/releases-2.9.3/tune/index.html`.

56. VERTA.AI. *VertaAI ModelDB* [online]. 2024-04. [visited on 2024-04-14]. Available from: `https://github.com/VertaAI/modeldb`.

57. BENTOML CONTRIBUTORS. *BentoML Documentation* [online]. 0004-2024. [visited on 2024-04-14]. Available from: `https://docs.bentoml.com/en/latest/index.html`.

58. DEEPCHECKS. *Deepchecks Monitoring Documentation* [online]. [visited on 2024-04-14]. Available from: `https://docs.deepchecks.com/monitoring/stable/getting-started/welcome.html`.

59. EVIDENTLY CONTRIBUTORS. *Evidently Documentation* [online]. 2024-04. [visited on 2024-04-14]. Available from: `https://docs.evidentlyai.com/`.

60. PYTHON SOFTWARE FOUNDATION. *What is Python? Executive Summary* [online]. [visited on 2024-03-15]. Available from: `https://www.python.org/doc/essays/blurb/`.

61. PALLETS. *Flask 3.0.x Documentation* [online]. 2024-02. [visited on 2024-03-15]. Available from: `https://flask.palletsprojects.com/en/3.0.x/`.

62. CHESNEAU, Benoit. *Gunicorn 21.2.0 Documentation* [online]. 2023-07. [visited on 2024-04-08]. Available from: `https://docs.gunicorn.org/en/21.2.0/`.

63. SOLEM, Ask; CELERY CONTRIBUTORS. *Celery 5.3.6 Documentation* [online]. 2023-11. [visited on 2024-03-15]. Available from: `https://docs.celeryq.dev/en/v5.3.6/`.

64. BROADCOM. *RabbitMQ Website* [online]. 2024-03. [visited on 2024-03-16]. Available from: `https://www.rabbitmq.com/`.

65. THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *About PostgreSQL* [online]. [visited on 2024-03-16]. Available from: `https://www.postgresql.org/about/`.

66. MALEKI, Sasan; TRAN-THANH, Long; HINES, Greg; RAHWAN, Talal; ROGERS, Alex. Bounding the Estimation Error of Sampling-based Shapley Value Approximation With/Without Stratifying. *CoRR*. 2013, vol. abs/1306.4265. Available from arXiv: `1306.4265`.

67. REGALADO, Antonio. *Who Coined 'Cloud Computing'?* [online]. 2011-10. [visited on 2024-04-16]. Available from: `https://www.technologyreview.com/2011/10/31/257406/who-coined-cloud-computing/`.

68. SUSNJARA, Stephanie; SMALLEY, Ian. *What Is Cloud Computing?* [online]. 2024-02. [visited on 2024-04-16]. Available from: `https://www.ibm.com/topics/cloud-computing`.

69. CNCF. *Kubernetes Project Journey Report* [online]. 2023-06. [visited on 2024-04-17]. Available from: `https://www.cncf.io/reports/kubernetes-project-journey-report/`.

70. THE KUBERNETES AUTHORS. *Kubernetes Documentation* [online]. 2024-04. [visited on 2024-04-17]. Available from: `https://kubernetes.io/docs/home/`.

71. HASHICORP. *Terraform Documentation* [online]. 2024-04. [visited on 2024-04-17]. Available from: `https://developer.hashicorp.com/terraform/docs`.

72. HELM AUTHORS. *Helm Documentation* [online]. 2024. [visited on 2024-04-18]. Available from: `https://helm.sh/docs/`.

73. THE KUBERNETES AUTHORS. *Kustomize Documentation* [online]. 2024-04. [visited on 2024-04-21]. Available from: `https://kubectl.docs.kubernetes.io/references/kustomize/`.

74. GITLAB. *Get started with GitLab CI/CD* [online]. 2024-02. [visited on 2024-04-22]. Available from: `https://docs.gitlab.com/ee/ci/`.

75. OUTERBOUNDS. *A minimal viable Metaflow-on-Azure stack* [online]. 2023-05. [visited on 2024-04-27]. Available from: `https://github.com/outerbounds/metaflow-tools/tree/b1af070934882bce679be107027c2aeeafbeee82/azure/terraform`.

76. INGRESS-NGINX CONTRIBUTORS. *Ingress-Nginx Controller Documentation* [online]. 2024-02. [visited on 2024-04-23]. Available from: `https://kubernetes.github.io/ingress-nginx/`.

77. THE CERT-MANAGER AUTHORS. *cert-manager Documentation* [online]. 2024-04. [visited on 2024-04-28]. Available from: `https://cert-manager.io/v1.13-docs`.

Appendix A

Acronyms

API	Application Programming Interface.
AWS	Amazon Web Services.
CACE	Changing Anything Changes Everything.
CD	Continuous Delivery.
CI	Continuous Integration.
CNCF	Cloud Native Computing Foundation.
CRISP-DM	CRoss Industry Standard Process for Data Mining.
CRISP-ML(Q)	CRoss-Industry Standard Process model for the development of Machine Learning applications with Quality assurance methodology.
DAG	Directed Acyclic Graph.
DevOps	Development and Operations.
GCP	Google Cloud Platform.
HTTP	Hypertext Transfer Protocol.
IaC	Infrastructure as Code.
LFS	Large File Storage.
ML	Machine Learning.
MLOps	Machine Learning Operations.
MVP	Minimum Viable Product.
OCI	Open Container Initiative.

SHAP SHapley Additive exPlanations.

SQL Structured Query Language.

UI User Interface.

VM Virtual Machine.

WSGI Web Server Gateway Interface.

Appendix B

Contents of the Archive

```
README.md .................. information about the contents of the archive
├── src ........................... directory containing the implementation
│   ├── .devcontainer .............. directory for the development container
│   ├── mlops....directory containing the pipeline implementation and scripts
│   ├── .gitlab-ci.yml ................... GitLab CI/CD pipeline definition
│   └── README.md .................... information about the implementation
└── thesis ........................ directory containing the text of the thesis
    ├── assets..............directory containing the assets used in the thesis
    ├── chapters ................ directory containing the text of the chapters
    ├── bibliography.bib......................................bibliography
    ├── FITthesis.cls..................LaTeX template for theses at FIT CTU
    ├── main.tex .................................... main LaTeX source file
    └── thesis.pdf ................................................ this PDF
```