Technical and organizational challenges when adopting DevOps principles for the integration of Artificial Intelligence into classical Software Engineering

Bachelor Thesis

for the degree of

Bachelor of Science

from the Course of Studies Applied Computer Science

at the Cooperative State University Baden-Württemberg Mannheim

by

Pascal Schroeder

Time of Project:	01/07/2019 - 23/09/2019
Student ID, Course:	5501463, TINF16AI-BC
Company:	IBM Deutschland GmbH, Ehningen
Supervisor in the Company:	Steffen Krause
Reviewer in Corporate State University:	Prof. Dr. Holger Hofmann

Declaration of Sincerity

Hereby I solemnly declare that my project report, titled *Technical and organizational challenges when adopting DevOps principles for the integration of Artificial Intelligence into classical Software Engineering* is independently authored and no sources other than those specified have been used.
I also declare that the submitted electronic version matches the printed version.

Mannheim, 23rd September, 2019

Pascal Schroeder

Contents

List of Figures

List of Listings

Abkürzungsverzeichnis

AI Artificial Intelligence

IT Information Technology

DevOps Development and Operations

API Application Programming Interface

REST Representional State Transfer

SaaS Software as a Service

ML Machine Learning

VM Virtual Machine

CPU Central Processing Unit

GPU Graphics Processing Unit

IDE Integrated Development Environment

ANN Artificial Neural Network

CRISP-DM Cross Industry Standard Process for Data Mining

LFS Large File Storage

CNTK Microsoft Cognitive Toolkit

UI User Interface

SDK Software Development Kit

JSON Java Script Object Notation

1 Introduction - Software Development Trends

The upcoming trends of Artificial Intelligence (AI) and Cloud Computing are changing the world of Information Technology (IT) as a whole. One very important aspect is the way applications are developed and operated. These processes can be summarized under the term Development and Operations (DevOps). This work will deal with how AI and Cloud force DevOps to transform and will discuss different approaches to adopt and realize those principles for AI applications. The following chapter will introduce the terms of AI and Cloud computing and will describe how these innovations force DevOps to transform.

1.1 Changes in Software Development caused by AI

First discussed in the 1940s, Artificial Intelligence has made great progress in recent years thanks to advances in computing capacity as well as Deep Neural networks and the accessibility of huge amounts of data. [1] This leads to companies investing in AI about 32€ Billion in 2019 according to a forecast by IDC, which makes it one of the most important fields of businesses. [2]

These progresses do not only impact the products itself, but also their development. In contrast to a traditional software development process in which the business logic is explicitly encoded in rules, solutions that rely on Machine Learning (ML) are fed with huge amounts of data that contains the business logic only implicitly.

This leads to a completely different development cycle. While classical software development is focused on the design and the development of the code, which is followed by testing and deployment, the machine learning lifecycle consists out of data collection, preparation, the training of the model with those data as well as the deployment of this model. [3] The differences between both is described in more detail in chapter 1.3.

This new type of software development is getting more and more important. Some people like Andrej Karpathy, Director of AI at Tesla, even predict, that these new type of software will replace traditional software development as a whole. [4] This has some advantages, e.g. that it is easier to learn and to manage, more homogeneous and very well portable. In return it is harder to understand for humans so that those systems appear to us as “black boxes”. Also debugging can be very difficult as well because usually there is no real error message. [3]

In this work, all the progresses that have led to this transformation will be explained in more detail. Additionally, it will be described which difficulties and challenges come along with it, focused on necessary changes for operations of the development cycle, called DevOps.

1.2 Cloud Computing

In 2009 the university of Berkeley published a paper, in which the potential of Cloud Computing has been discussed. In doing so Cloud Computing has been defined as both - the applications delivered as services over the Internet, referred to as Software as a Service (SaaS), as well as the hardware and the systems software in the datacenters that provide those services. [5]

Thereby a distinction is made between a Public Cloud and a Private Cloud. A Cloud is called public when it is made available in a pay-as-you-go manner to the public. This means, that you pay for a service in advance and then you can only use what you have paid for. Private Cloud on the other hand refers to internal datacenters of a business or other organizations unavailable to the public. [5]

SaaS in general has advantages for both end-users as well as service providers. While the service provider can install, maintain and control their services more easily, the user can

access the service anytime, anywhere, collaborate more easily and keep their data inside the infrastructure. [5]

Cloud Computing enables this possibility to more application providers and additionally also the possibility to scale their applications on demand. Based on this possibility there are some more advantages identified for Cloud Computing in particular:

- The illusion of infinite computing resources on demand
- The elimination of an up-front commitment by Cloud users
- The ability to pay for use of computing resources [5]

The first point eliminates the need to plan far ahead how many resources are needed but enables the user to buy as many resources as necessary as soon as it is needed.

Also, Cloud users are allowed to change their need for resources any time, so they can scale up their application and don't have to commit to their need beforehand as described in point two.

The last point is about saving money when the resources are no longer needed. As it is possible to scale up, it is also possible to scale down, which eliminates the need to pay for the resources. This relieves the Cloud users from taking too much risk when paying for resources they may not need for a long time.

For realizing those advantages different approaches were competing with each other: One looking similar to physical hardware with the ability to control the entire software stack similar to a low-level Virtual Machine (VM). The alternative was a clean separation between a stateless computation tier and a stateful storage tier. In this competition, the first option has become established, because in the beginning most users wanted to recreate their local environment instead of writing new programs solely for the cloud. [6]

But this solution still forced the User to manage their environment themselves. Especially for simpler applications, it is desirable to have an easier path to deploy their applications to the Cloud.

Amazon provisioned a new solution for those needs in 2015 called AWS Lambda. Lambda offered the possibility to simply write the code to be executed and leaves all server provisioning

and administration tasks to the cloud provider. [7]

This is known as serverless computing, because - even though there are still servers being used for the computation tasks - the user does not have to care about any tasks on serverside and can focus on writing the code. Serverless services must scale and bill automatically based on usage without any need for explicit provisioning. [6]

Those new possibilities with the Cloud technology changed the way of how developers can deploy and manage their products. Additionally, also AI development benefit from this very strongly because of several reasons.

The first one is automated scalability, which eliminates the need to worry about raising workloads. Additionally, when AI models are being trained, there are many resources needed, but besides this task, there are not as many resources necessary. Through serverless computing, this idle time will not be billed. [8]

Serverless computing also reduces the interdependence, because the functions can be updated, deleted or invoked any time without having any side effect on the rest of the system. [8]

All this forced developers to change their way of developing and operating their products. This change of the DevOps lifecycle will be introduced in chapter 1.3

1.3 Development Operations in time of AI

The traditional software development has continuously been improving for years, defining standards and principles to increase the efficiency, portability, and quality of the development cycle. Those principles and practices are called DevOps. In the new type of software development described in chapter 1.1, not all of those are applicable any more and new ones need to be defined. Additionally, the upcoming Cloud technology changed the way how products can be deployed and managed as described in chapter 1.2. In this chapter, the consequences of those changes for DevOps will be examined.

Starting with development itself and ending with maintaining and improving the product,

there are DevOps principles for every single stage of the development cycle. This is described in more detail in chapter 2.1. In the new world of AI and Cloud, those stages are different, some new stages have been added, others have become unnecessary.

A short, simplified comparison between the required steps of traditional operations and Machine Learning operations can be seen in figure 1.1.

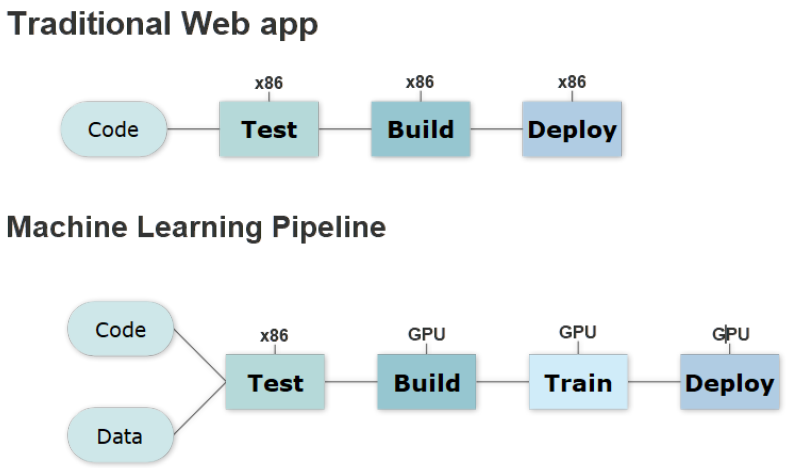


Figure 1.1: Comparison development lifecycle traditional app and ML pipeline[9]

In traditional development, there is one input - the code. This code has to be tested, built and then deployed.

The Machine Learning pipeline looks different. First, there is an additional input - data, which is very important for Machine Learning applications. The code and the data still need to be tested afterward.

Also, it needs to be built. The difference in this stage is, that the resources used for building an ML application are usually different. While traditional applications are usually built on regular CPUs (Central Processing Unit), Machine Learning pretty much rely on GPUs (Graphics Processing Unit), which leads to a more heterogeneous and complex hardware landscape.

A third difference is the training step. ML applications have to be fed with training data. This step can take very long especially when insufficient hardware is being used.

The last differentiator is, that ML pipelines are continuously updated with new data. This process is called “refitting” the model.

Besides the rise of AI, Cloud Computing also changed the way of how products are deployed and managed. The Cloud can serve as a centralized platform for testing, deployment, and production. Also, it can automatically scale applications and allocate needed resources. Serverless computing even eliminates the need to set up a complete system and offers a simple way to deploy Machine Learning or other functions. [10]

AI tools on the other hand are enabling new possibilities for improving the IT operations functions like monitoring, analysis, service management, and automation.

All those progresses led to two big trends in the DevOps area. The first one is using Artificial Intelligence for DevOps. In doing so Machine Learning functionalities are used to enhance all IT operations. Gartner calls this “AIOps” and predicts, that the use of AIOps will rise from 5% in 2018 to 30% in 2023. [11]

The second trend is using DevOps for the development of AI. This means adopting existing principles and practices for the traditional development cycle to the development cycle of AI tools. The objective of this is to upheld existing standards and maximize the efficiency of the development processes. [9]

In this work this second trend will be analyzed, the influences leading to it will be described and possible solutions will be shown, compared and discussed.

2 Theory - DevOps, Cloud and Machine Learning

In this chapter, the theoretical basis, that is needed for creating DevOps principles for AI, will be given. For that, it will be started with an explanation of what DevOps is in general. Then it will be shown, which new possibilities came with Cloud and 12-factor-apps in chapter 2.2. After that, the basics of Machine Learning and AI will be given in chapter 2.3, specializing on the lifecycle of AI development in chapter 2.4. Last in chapter 2.5, all these knowledge will be combined to adopt the DevOps principles explained in chapter 2.1 to the new world of AI with the help of Cloud technologies.

2.1 Development Operations

Since people started manufacturing products on a mass scale, the goal is to increase the efficiency of this manufacturing process and reduce waste of time and material.

One early set of best practices for manufacturing was the concept of *Lean manufacturing*, which tries to reduce the waste of resources and time of a production cycle as much as possible. With the upcoming use of software as a commercial product in the 1970s [12] a desire came on to create best practices for developing and operating products the same way as it was already usual in conventional manufacturing. [13]

In 2009 two Flickr employees - John Allspaw and Paul Hammond - presented their way of combining Development and Operations. Inspired by this presentation, a Belgian consultant

named Patrick Debois formed a new conference - the “Devopsday” in Ghent. This naming is how the term “DevOps” has been created and prevailed. [14]

Since then, DevOps has been established or at least planned in 91% of all companies as an essential way to increase their efficiency of software development. [15] For almost every stage of development, there are principles and practices defined and continuously improved. However, before those practices are explained, further insight into the business need will be given. All this will be oriented to the book “DevOps for Dummies” by Sanjeev Sharma and Bernie Coyne. [16]

Every process or product need a business value that covers the costs caused by it. For that, there must be either an outcome for the customer or reduced producing costs.

For DevOps, it is usually even both - on the one hand, enhanced customer experience can be guaranteed, and on the other hand, the efficiency of the production cycle can be increased.

One example of enhanced customer experience are practices to get fast feedback from all stakeholders. This feedback can then be used to improve the designed product. One of such practices is the so-called “A-B testing”. There, two different sets of features are enrolled in two groups of randomly chosen users. Both can give their feedback to the producer, and the set with better feedback will then later be enrolled for every user.

The efficiency can be increased through reduced waste and rework with practices to write reusable components. Other examples are tools for planning a product or fast ways to deliver a product without the need to redeploy everything step by step. In this chapter, the advantages of DevOps will be delighted in more detail, and some of the practices will be described and explained.

2.1.1 Principles

The DevOps movement is generally based on four principles. The first one is to *develop and test against production-like systems* to move operations concerns earlier in the life cycle. The purpose of this is to see how the system behaves and performs before it gets deployed. This is also advantageous from an operations perspective, because it can be seen, how the system

behaves when it supports the application.

The second principle is to *deploy with repeatable and reliable processes*. The objective is to create a delivery pipeline, that enables continuous, automated deployment and testing of the product.

Third, it is crucial to *monitor and validate operational quality.* This means that applications and systems should not only be monitored in production, but already earlier. This forces automated testing to be done early and often to monitor the application. Metrics should always be captured and analyzed to provide an early warning system about potential issues and risks.

The last principle is to *amplify feedback loops* intending to enable a quick reaction to issues. For this, organizations need to create a communication channel to their users, so that they can give feedback, and the developers can react to it accordingly.

2.1.2 Practices

The DevOps practices that have become commonplace can be split into four different sets based on the different periods of a product lifecycle. To each set, there are several practices, standards, and tools available, which help to achieve the best possible result. The four different sets and some example practices can be seen in figure 2.12.

The first set is called **Steer**, which is about managing and planning the development and lifecycle of a product. This set includes establishing and continuously adjusting business goals. The process of resolving this issue is called *Continuous Business Planning*.

This practice includes three major points - the acceleration of software delivery, a good balance between speed, cost, quality and risk, and the reduction of the time it needs to get customer feedback. [17]

These points are mainly fulfilled via tools and practices to track the status, feedback, and needs of a project efficiently. First, a vision of the projects overall objective should be created, and every action should be guided by it. [17]

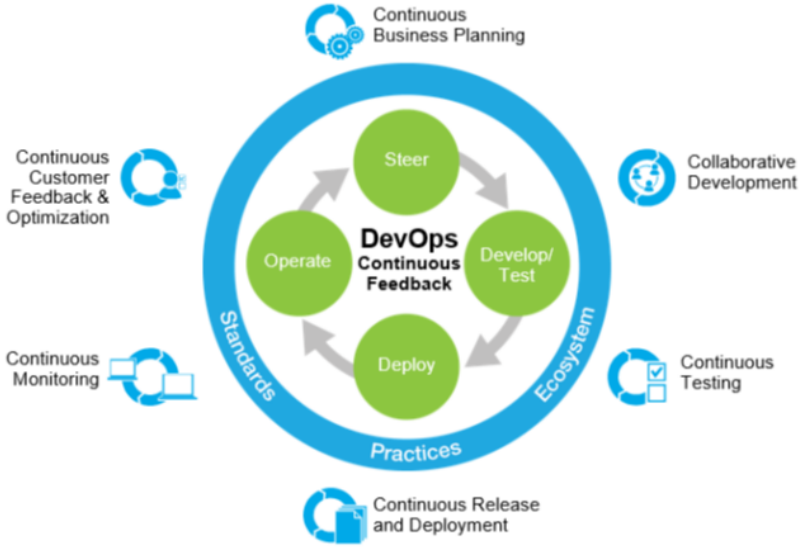


Figure 2.1: DevOps reference architecture[16]

The strategy, which should lead to this vision, has to be monitored and adjusted continuously based on new information and feedback. This procedure is called *continuous improvement*. For that, a good dialog between a Business and IT is necessary for defining good scopes and priorities. [17]

Then a good plan can be built, for example with a release roadmap, which determines which feature should be completed at what time. This approach is called *release planning* and helps to track the progress of the project and makes it easier to react on new trends, feedback or issues and adjust the single steps based on this. The status of each release, as well as every single feature, has to be continuously tracked so that risks will be recognized as early as possible to increase the available time to react. [17]

The second set of practices is for the time of **development and testing**. Two eminent practices for this are *collaborative development* and *continuous testing*.

Collaborative development enables different practitioners - architects as well as analysts, developers, specialists, and other participants - to work together on one project. For that, it provides a standard set of practices and a common platform to create and deliver the software.

One core capability is a practice called *continuous integration*, in which developers continuously or frequently integrate their work with the other developers. For that, a shared platform

or repository is necessary, on which the developers can frequently commit their changes in the code. Often, this is done using a version control system like Git, which does not only enable version controlling and continuous integration, but supports almost the whole operation lifecycle, which is known as *GitOps*. *GitOps* is a special kind of DevOps and starts with a Collaboration platform like Github to enable a group of people to work together. It also enables version controlling and continuous integration. What a typical integration workflow with Git looks like can be seen in figure 2.2.

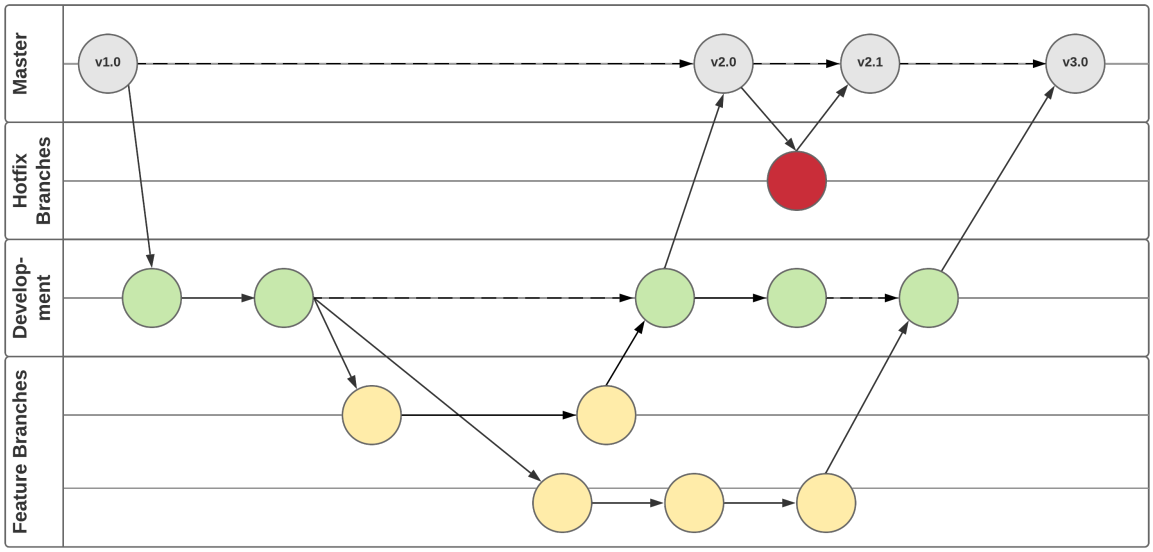


Figure 2.2: Git workflow

There the project is split into several branches, which are represented by a rectangular box. The circles represent single commits of new code. Arrows are representing the push of a new version and dashed arrows are representing the base version, on which the new commit was built and to which it will be merged.

The most important branch is the master branch. On the master branch, every finished version is pushed, and from this branch, it can be released. The development branch is for ongoing changes. Small changes can be done directly on the development branch. More significant changes, like added features, should get their branch. Every developer can create their branch for a new version so that the developers do not interfere with each other. When all code changes are done, the new version will be merged into the development branch. Sometimes

some conflicts in the code have to be fixed for a clean merge in case two different commits changed the same code piece. As soon as the new version is working on the development branch and everything is tested and ready, it can be merged into the master branch, so that this new version can be rolled out. In case an error occurs, it can be fixed in a hotfix branch directly descending from the master branch.

The master branch can then serve as trigger for a delivery pipeline, which compiles the code and moves it to the deployment environment, for example a Kubernetes cluster. This fianlizes the *GitOps* cycle, which shows, that a *GitOps* workflow can support the development from coding to deployment.

Additionally, the application should be tested and verified continuously. For that, the developer can run local unit tests to verify their changes before integrating. Unit tests test a specific component with defined input and output and checks if the calculated output is the expected one. However, this does not verify that the integrated code performs as designed. [18] A continuous integration service like Jenkins can relieve the developer of this task and automatically builds and runs unit tests on the newly committed and integrated code. In doing so, it runs not only unit tests, but also integration tests, which test the software as a whole. This process is called *continuous testing*.

Another critical point is to shorten the delivery cycles through an end-to-end integration so that it needs less time to enroll a new feature or similar. This method leads to quickly given feedback and enables a faster reaction to this. [17] This approach takes place in the **Deployment** stage of a product lifecycle and one of the root capabilities of DevOps. It deals with the automation of the deployment of the software to the different environments, which is called *continuous delivery*.

After the deployment follows the **Operation**. During this stage, the performance of an application should be monitored, and feedback should be collected. The results of this should be used to improve the product as well as other products, that will be developed in the future. For this, there are two practices defined - *continuous monitoring* and *continuous feedback and optimization*.

Continuous monitoring provides data and metrics to the performance of an application as well

as its running server, the development cycle, the production, and other stakeholders.

Continuous feedback, on the other hand, provides data coming directly from the customer. This data includes the behavior of the users as well as feedback provided by them.

Based on those retrieved data, businesses may adjust their plans and priorities, improve the development cycle and features, and enhance the environment in which the application is deployed in a more agile and responsive way. The objective of this is to improve the product and satisfy the users. Additionally, this knowledge should be used for new products, that will be developed in the future.

2.1.3 Technologies

One technology to allow developers to follow above practices is **Infrastructure as code**, which enables organizations to deploy their environments faster and on a larger scale. This technology is implemented with machine-readable definitions and configurations. Based on them, the machine can provide the necessary environment automatically to enable continuous delivery.

One of the most important technology for DevOps are **delivery pipelines**. A delivery pipeline controls the product cycle of an application from development to production. Typically there are four or more stages - development, test, stage, and production. This can be seen in figure 2.3.

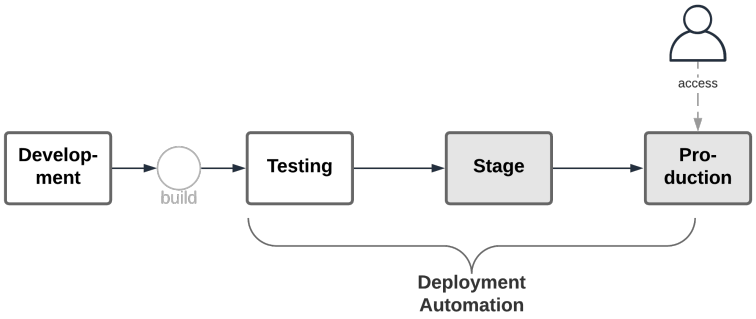


Figure 2.3: Delivery pipeline[16]

For every stage, there is usually one specific environment. Those are represented as a box.

The dark boxes represent a production like an environment.

In the development environment, all the code updates are being done. There are tools provided like IDEs (Integrated Development Environment) to write the code as well as tools that enable collaborative development like source control management or project planning.

Source control management is typically combined with version control. This enables the developer also to store previous versions of his application and reduces the risk of issues in new updates because it can roll them back in case of an error.[19]

After the development, the delivery pipeline must care for the application to be built.

Second, there is the testing environment, in which single components can be tested. For that, it has to manage test data, scenarios, scripts, and associated results. Similar to the development environment, it must not look like the production environment.

The next one is the staging environment, in which the system can be tested as a whole. The staging environment should be as much production-like as possible so that as many as possible required services, databases, and configurations can be connected and applied without touching the production environment. The stage environment is for testing the application before rolling out a major update. [20][21]

The last one is the production environment, in which the application will be running live and accessible for the users.

The delivery pipeline consists of all those stages and manages an automated transition from one stage to the next one starting directly after the development.

For this deployment automation tools are necessary, which perform the deployments and track which version is deployed on which node. It also manages changes that need to be performed for the middleware components and configurations as well as database components or the environment.

Last, there should also be a tool for **release management**, which provides a single collaboration portal for all stakeholders participating in a project to plan and track the releases of an application and its components across all stages.

With such technologies, most of the defined practices can be performed with the help of accordingly educated people and well-thought-out processes. However, DevOps is no static set of practices and tools, but it changes with the changes in the world IT, such as Cloud or AI. In the next chapters, these technologies will be reflected, and the impact those have on DevOps will be analyzed.

2.2 Microservices and 12 factor apps

As described in chapter 1.2 Cloud opened new possibilities of deploying and maintaining software. This eases the deployment itself as well as rolling updates without any downtime. Additionally, it enables high scalability as well as high availability.

One model of Cloud Computing is Software as a Service (SaaS). In this model, an application is deployed on the provider's platform and is accessible via the internet on demand. This way, the end-user can access the software from anywhere anytime. In doing so, there is no need to deploy, install, or maintain anything. The user has to pay-per-use, which means that he only pays for the resources, which he has claimed. [22]

However, the development and deployment of portable, resilient applications that will thrive in Cloud environments are different from traditional development. Because monolithic applications need to be rebuilt entirely as soon as one component is being changed, the development is unflexible. Additionally, the scaling of a single component needs a scaling of the whole application. Both are disadvantages which are opposed to the new possibilities a Cloud deployment offers. The solution was to build those applications not as one monolithic software but as a suit of services.

2.2.1 Microservice Architecture

For that, the term Microservice Architecture has sprung up over the last few years. There is no unique definition for it, but when talking about Microservice Architecture mostly, it is referred to Martin Fowler's characteristics described in [23].

Following Fowler, the first characteristic is, that componentization is realized via services. In monolithic software, different components are linked together via libraries. Services on the other side are out-of-process components. The communication is realized with web service requests or remote procedure calls. The advantage of this approach is that they are independently deployable, which is the reason why this is the usual approach in Microservice Architecture.

A second characteristic is that Microservices are organized around business capabilities. This means that the services take a broad-stack implementation of software for that business era. This also leads to cross-functional teams, which are working together on building the Microservice.

Additionally, Microservices are handled as products instead of projects. The difference is that while products are supported and owned by the product team over its full lifetime, projects are not. They are completed as soon as a specified set of functionalities is implemented. After an operation team would support those projects. The approach to treat Microservices as products has the advantage, that the development team has full responsibility for it and are probably more interested in a clean, well functioning long-term solution.

Another characteristic is the use of smart endpoints and dumb pipes, which means that the services are as decoupled as possible. The only way they stick together is that they receive requests of each other and produce responses after applying the defined logic. This communication is handled via simple RESTish (Representional State Transfer protocols.

Also, the governance of Microservices is usually decentralized so that every service can be built on different technology platforms, and there is a different approach to standards. This decentralization leads to a more flexible environment.

The same applies to data management, which is decentralized as well. Each service can manage its database, which avoids problems through different conceptual models. While in centralized data management changes are usually made via transactions, in a decentralized data management system, there is transactionless coordination between the services. This does not necessarily result in consistency, but this cost is less than the cost, that would come up if a consistency would be forced to Microservices by distributed transactions. Instead, the

problems this eventual consistency could cause are dealt with by compensating operations.

The next characteristic is the automation of the infrastructure. This includes automated testing and deployment. It is essential to test every single Microservice intensively because the operational landscape for each can be strikingly different. Also, the deployment could differ from service to service.

Important for Microservices is that they are designed for failure. This means that it detects failures quickly and automatically restore it in case an error happens. This approach demands real-time monitoring. To ensure this functionality, some companies are even executing tests in production by purpose to observe the behavior of the system if one service fails.o

The last characteristic defined by Fowler is the evolutionary design. This design enables a more granular release planning because every change can be handled as a single and only modified services needs to be redeployed. These changes should not affect the communication with other services, which is the reason why they should always be designed as tolerant as possible.

The advantage of those Microservices for Cloud applications is mainly caused by possibilities to treat every component as a single. This improves the scalability, productivity, and the speed of the application as well as the development. This is caused by the faster way of developing such independent services. Additionally, it is easier to maintain services instead of a monolith application, and it gives more flexibility in technologies. Still, the development of Microservices should follow some guidelines to support the concept of independently managed and iterated services.

2.2.2 12 factor apps

One standard set of guidelines and best practices for the development of Cloud-based software and especially Microservices are the 12 factors drafted by developers at Heroku. These factors are

- Codebase - For every deployed service there should be precisely one codebase, for example, an IT repository
- Dependency - Services should explicitly declare and isolate all dependencies

- Config - Configurations for the deployments should be stored in the environment
- Backing services - All backing services should be treated as attached resources
- Build, release, run - The delivery pipeline should be strictly separated into building, releasing and running
- Processes - Apps should be executed in one or more stateless processes
- Port binding - Services should be exposed by listening on a specific port
- Concurrency - Concurrency is achieved by horizontal scaling
- Disposability - The objective should be a robust and resilient system with fast startup and graceful shutdown
- Dev/prod parity - The development, staging, production and every other environment should be as similar as possible
- Logs - Applications should produce logs as event streams
- Admin processes - Admin tasks should be packaged alongside the application to ensure that it is run with the same environment [24]

Following these guidelines, stable and performant Microservices can be built. In the last years, some technologies have emerged as particularly suitable for developing such services.

2.2.3 Container - Docker

One of them is containerization of applications. This can be understood as a package for the isolation of application within a closed environment, which provides everything the application needs. It is comparable to a VM (Virtual Machine), but much more light-weighted. [25] This enables a light deployment without unnecessary services or applications running in the background, which leads to a very performant execution.

An industry-leading container engine technology is Docker. In figure 2.4 the differences between a VM and Docker can be seen.

On the left side, the infrastructure of a VM can be seen, on the right side, the infrastructure of a Docker container. Both need the infrastructure of a physical device and its host operating system. On top of a VM on this Host Operating System, there is a Hypervisor, and on this

Hypervisor several Guest OS can be running. On those again, the apps itself can be executed, and the necessary libraries and binaries are running in the background.

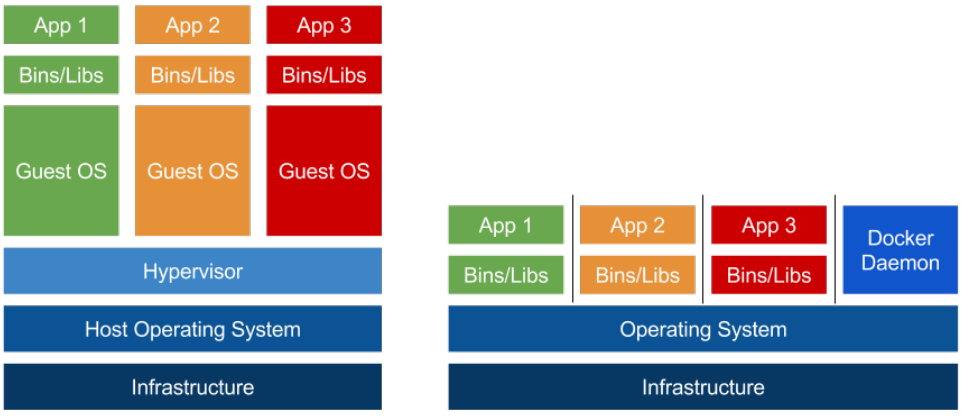


Figure 2.4: Comparison between Docker and VM[26]

In case of a docker container, those binaries and libraries are directly running inside of a container on the operating system. There is no need for a hypervisor or a complete version of a Guest OS. This approach also enables the app to be running on top of that. The containers are isolated from each other in different namespaces and own network stacks. This means that processes running within a container cannot see or interact with processes of other containers, and they do not get privileged access to sockets or interfaces of other containers. [27]

Additionally, a Docker Daemon is running in another process. The Docker Daemon has three main tasks - listening and processing API requests from the Docker client to run Docker commands, managing Docker objects (images, containers, volumes, and networks) and parsing Dockerfiles for building Docker images. [28]

With this technology, several of the 12 factors described above are fulfilled. One factor is the explicitly declared and isolated dependency management. Within the Dockerfile, every dependency needs to be explicitly declared to fulfill all the requirements of the application. [29]

Also, Docker containers cannot communicate with each other directly but need to communicate externally over with backing services over the network [29]

Additionally, Docker containers are executed as stateless processes with ephemeral storage only. [29]

The development and production parity is given because containers standardize the way of delivering applications as well as its dependencies. [29]

Even admin processes can be run as one-off processes inside the Docker container through jumping inside the container and executing all necessary commands.

Still, in a local environment, not each of the 12 factors is fulfilled. Instead, an enabler is needed, which scalable and failure safe way to deploy those containers.

2.2.4 Kubernetes as an enabler

Kubernetes can serve as such an enabler. It can host Microservices as Docker containers and ensure all of the 12 factors to be met.

In general, Kubernetes enables an automated deployment, scaling, and management of these containers within a cluster of nodes. Thereby a cluster consists of at least one master node and any number of worker nodes. Figure 2.5 shows the different services owned by master and worker nodes.

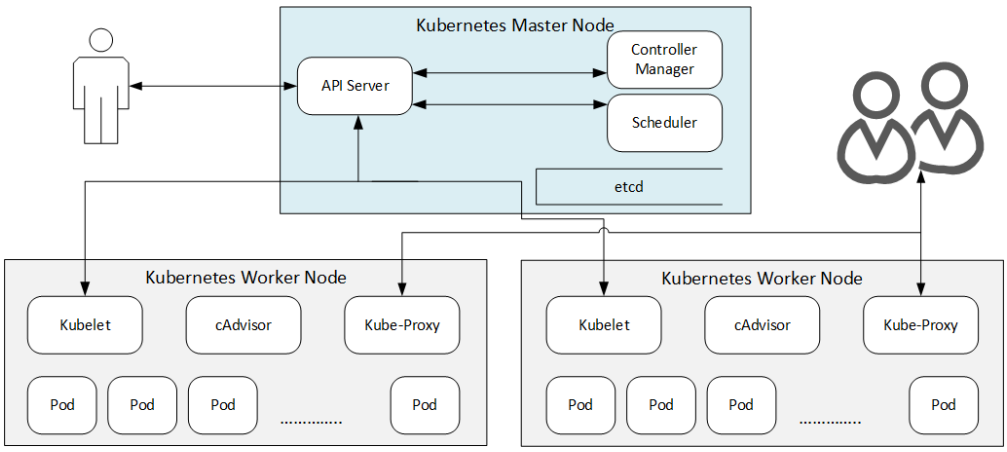


Figure 2.5: Kubernetes service allocation[30]

First, there are several pods on each worker node. Pods are the smallest unit in Kubernetes. They contain one or more containers, which are deployed together on the same host. There

they can work together to perform a set of tasks. [31]

On the master node, there are an API (Application Programming Interface) Server, a Controller Manager, a Scheduler and a key-value store called etcd. [30][32]

The API Server is for clients to run their requests against it. That means the API Server is responsible for the communication between Master and Worker nodes and for updating corresponding objects in the etcd. [30][32]

The Controller Manager is a daemon, which embeds all of the Kubernetes controllers. Examples for them are the Replication Controller or the Endpoint Controller. Those controllers are watching the state of the cluster through the API Server. Whenever a specific action happens, it performs the necessary actions to hold the current state or to move the cluster towards the desired state. [30][32]

The scheduler manages the binding of pods to nodes. Therefore it watches for new deployments as well as for old ones to create new pods if a new deployment is created or recreating a pod whenever a pod gets destroyed. The scheduler organizes the allocation of the pods within the cluster based on available resources. [30][32]

The etcd is a key-value store, which stores the configuration data and the condition of the Kubernetes cluster. [30][32]

The worker node consists of a Kubelet, a cAdvisor, a Kube-Proxy and - as mentioned before - several Pods.

The Kubelet needs to be used if a new pod should be deployed. Then it gets the action to create all needed containers. For that, it uses Docker to create them. Afterward, it combines some containers into one pod. Containers in one pod are always started and stopped together. This pod will then be deployed on the node, on which the Kubelet is located. [30][32]

The cAdvisor measures the usage of CPU-resources as well as demanded memory on its node. That information is forwarded to the master node. Based on those measurements, the scheduler allocates the pods within the cluster to ensure the best possible allocation of resources. [30][32]

The kube-proxy is a daemon that runs as a simple network proxy to provide the possibility of communicating to that node within the cluster.[30][32]

With this architecture, Kubernetes enables all the factors that are missing in a local deployment of Docker containers.

First, the codebase of the deployment is given as YAML or JSON file and the container in Dockerfile. This way, source control of all the necessary code can be done using git, for example. [33]

Also, the dependencies for one Microservice can be checked easily with the functions *readinessProbe* and *livenessProbe*. While the *readinessProbe* tests whether there are backing services, the *livenessProbe* tests if the backing services are all healthy. In case of a missing or failed Microservice, the appropriate pod is automatically restarted. [33]

For storing all the necessary configurations in the process environment table, Kubernetes provides ConfigMaps. With these, the containers can retrieve the config details at runtime. [33]

Stage separation is achieved through artifact management. Once the code is committed, a build occurs, and the container image is built and published to an image registry. These releases are then deployed across multiple environments. [33]

The port binding is guaranteed through Kubernetes services. These are objects to declare the network endpoints of a service and resolve endpoints of other services specified to a port of the cluster. [33]

The concurrency is a factor, which is handled especially extensively. It allows the services to scale at runtime depending on the replica sets defined in the declarative model. Also, Kubernetes has introduced autoscaling based on compute resource thresholds. [33]

Also, disposability is fulfilled because every pod can be destroyed or started quickly. Additionally, Kubernetes will automatically destroy unhealthy pods. [33]

Last, logs are written to *stdout* and *stderr* and can be easily accessed. They are not stored and managed as internal files. [33]

This way, Docker and Kubernetes are fulfilling each of the 12-factors, which shows that it is an excellent way to provide Microservices. This is the reason why many big companies decided to use Kubernetes as an enabler for big platforms like Google Cloud to give just one example.

2.3 Machine Learning

Another eminent movement in IT is Machine Learning, which helped AI to a new hype and opened several new possibilities and improvements to software development. Because of the exponential growth of computation power and data, Machine Learning has become one of the most important tools when it comes to Artifical Intelligence.

The advantage of Machine Learning compared to traditional software development is, that it eliminates the need to write the code by oneself. Instead, the developer enters some input data to the Machine Learning system. This system then figures out mathematical functions, which describes the given collection of data points best. The process of finding these function is called Machine Learning, and the resulting function is mostly referred to as model.

This results in a system, which continuously improves the more data it is fed with because this leads to a more accurate function. Based on this assumption, Tom Mitchell defined Machine Learning in 1997 as follows[34]:

Definition 1 *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at task in T, as measured by P, improves with experience E.*

This definition means that a computer program learns if it is improving its performance at some task through experience. This experience could be input data like pictures of trees, with which it gets trained with the task to identify those trees on that picture.

2.3.1 Tasks

For fulfilling such a task, it is necessary to define this task before. This definition is oriented to the book “Deep Learning” by Ian Goodfellow, Yoshua Bengio, and Aaron Courville. [35]

In general, tasks are described in terms of how the Machine Learning system should process an example. An example is a collection of quantitative measurements, called features. This set can be written as a vector $x \in \mathbb{R}^n$ with each x_i being one of the features.

The most important and most common tasks in computer science are

- Classification
- Regression
- Structured Output
- Anomaly detection
- Missing Values

In *classification*, the objective is to figure out the category of an example out of a data set by given features. Mathematically this can be described as a function, that takes an example with all its features. Then it assigns a category out of an appropriate set. This set of categories is limited to k so that it can be described as $1, ..., k$. This function can be represented as follows:

$$f : \mathbb{R}^n \to 1, .., k \tag{2.1}$$

An example of this is character recognition. It gets an image of a character as input. Then, the task is to assign this image to one element of the set of characters.

The objective of a *Regression* is to predict a numeric value by given input data. This can be formulated as

$$f : \mathbb{R}^n \to \mathbb{R} \tag{2.2}$$

When the function that has to be learned is of the form $y = f(x)$ with $y \in \mathbb{R}$ and $x \in \mathbb{R}^n$ then x should be a set of independent variables and y should be a dependent variable on x.

An example of a Regression is a prediction of the costs for a new apartment based on features like size, location, and the number of rooms.

Structured Output tasks ask a learning algorithm to figure out the probability distribution that

has produced the data set. This value is then used to give an estimation of the relationship between single examples from the data set.i

A well-known example of this is to recommend products based on previously bought products. For that, the learning algorithm figures out the similarity between the products and recommends those with the highest consistency.

Anomaly detection works similar to Structured Output, but instead of looking for elements with a great relationship, it uses the probability distribution to check for irregularities within the dataset.

This is an excellent way to anticipate a fraud in a banking account through checking every transaction and hold back those, which are looking too irregular.

For *Missing Values*, the probability distribution is used to infer what value should most likely be set at a specific position of a given, incomplete set of examples. This means that a set $x \in \mathbb{R}$ with some x_i missing. The objective is to figure out the best values for these missing entries.

An example of this is to predict a logical sequence of numbers without the need to know the exact formula of the sequence.

2.3.2 Training approaches

As mentioned above, in Machine Learning, all those tasks are not solved by explicitly coding algorithms and formulas, but by training a system with given input data. For training those systems, different approaches are existing, each used for different requirements. This chapter will also be based on the book “Deep Learning”. [35]

One of the viable approaches is called supervised learning. Supervised learning requires an additional value for each example representing the optimal solution for it. Formally the data set can be described as $\mathbb{D} \in 2^{\mathbb{R}^n}$. For every example vector $x \in \mathbb{D}$ there exists a $y \in \mathbb{R}$ such that $y = f(x)$ with $f : \mathbb{R}^n \to \mathbb{R}$ being the function that describes the task to be learned. This additional feature is mostly referred to as label or target. Usually, the tasks are accomplished by approximating the conditional probability distribution $p(x|y)$.

Supervised learning algorithms are often used for classification or regression tasks.

Unsupervised learning, on the other hand, does not have any additional features but has to work with the pure, raw data set. Usually, the data points have many features, and the task is to figure out some useful property about the relationship between these data points. This is usually done by using the probability distribution function $p(x)$ of the data set.

Semi-supervised learning uses both - unlabeled examples as well as labeled examples - to predict the outcome. This approach facilitates the task for the developer, because not every data point has to be labeled, but decreases the size of labeled examples, which could worsen the resulting model.

Common tasks for unsupervised learning are clustering or anomaly detection.

In figure 2.6, a comparison between supervised and unsupervised data sets used for clustering and classification can be seen. The data shown is not of a particular use case and just for demonstration purposes. In both representations, the examples of the data set have only two features, one corresponding to the x-axis and the other on the y-axis. The task is to identify clusters in the data.

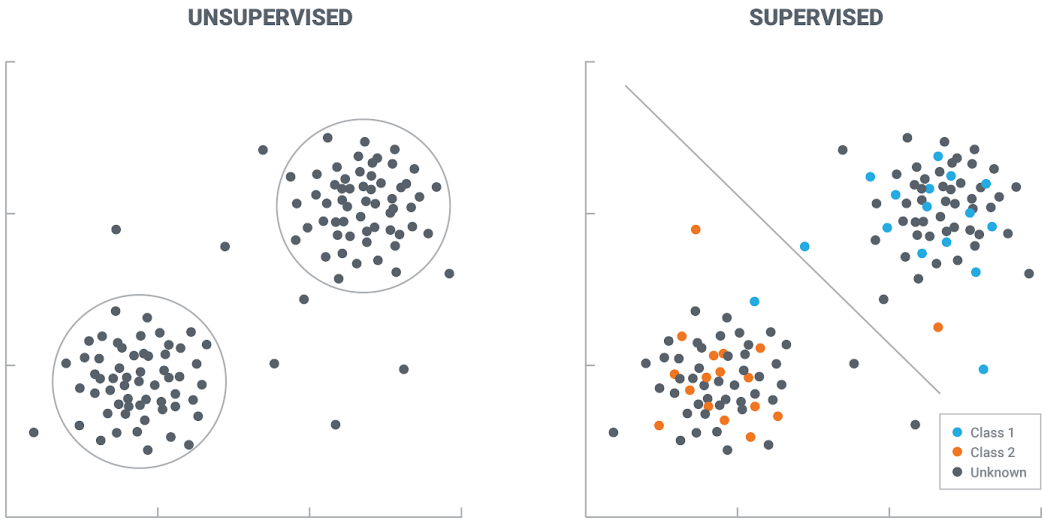


Figure 2.6: Comparison of unsupervised and semi supervised data sets used for clustering[36]

The difference is that some of the data points of the second data set are labeled with either "Class 1" or "Class 2". In the figure, this can be seen by the coloration of the data points. Even though in this example there are only some data points labeled, which is the reason why the applied learning approach is called semi-supervised learning, it is still functioning as an example for supervised learning and demonstrates the differences to the unsupervised model.

The missing labels force the first example to only use the features x and y and look for similarities to determine its association with a group. In the second examples, the algorithm can use the labels as well to determine a function, which assigns a class to every data point.

In the figure on the left side, two clusters can be seen, which are circled in a group. On the right side, there is a line, which marks the function splitting the dataset into two groups. It is noticeable that the first approach leads to some examples, which cannot be assigned to either of the classes.

Additionally, there is another approach called reinforcement learning. In reinforcement learning, a model is built and then iteratively improved by taking in further examples. For that, it needs to get some feedback on how good the built model is. For example, this can be a reward or punishment function.

2.3.3 Models

With the approaches mentioned above the objective is to create and train a model, with which the tasks can be fulfilled. In modern Machine Learning, the most important type of such models are Artificial Neural Networks (ANNs).

Neural Networks are inspired by biological neural networks like animal brains. According to that, ANNs are based on a collection of nodes called artificial neurons. Each neuron gets one or more input values and one output value. To generate the output value, the neuron does some mathematical computations with the input values. [37]

A network of such neurons usually persists out of at least three layers. The first layer is the input layer of the data, with which the neural network should be fed. Those values are then sent to every neuron of the hidden layer. Those are doing the necessary calculations described

below and send their output to the output layer. [37] This can be seen in figure 2.7.

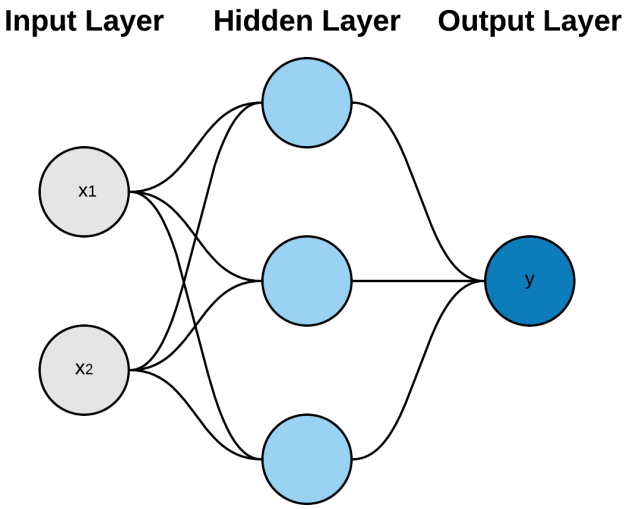


Figure 2.7: Simple neural network

On the hidden layer the neurons are combining the inputs and calculating a new value with them. For that three parameters need to be given to generate the output y with the input vector x:

- A weight vector w
- A bias b
- An activation function $f(x)$ [38]

The weight vector is used to weight the different inputs. This means, that each input is multiblied by a weight as can be seen in the equation below [38]

$$x_1 \rightarrow x_1 * w_1 \tag{2.3}$$

After that, all the weighted inputs are added together. Additionally the bias b is added to this formula: [38]

$$(x_1 * w_1) + (x_2 * w_2) + b \tag{2.4}$$

Finally, the activation function is being used to generate the output. For that it takes the sum calculated in the above equation as input: [38]

$$y = f((x_1 * w_1) + (x_2 * w_2) + b \tag{2.5}$$

A typical activation function is the sigmoid function, which outputs numbers in the range (0,1). The higher the input, the closer the result gets to 1. The lower the input, the closer it gets to 0. The sigmoid function can be seen below [38]

$$y = \frac{1}{1 + e^{-x}} \tag{2.6}$$

In figure 2.8 this whole process, that happens within the artificial neuron, can be seen in a simplified way. It takes the inputs x_1 and x_2, multiplies them with their belonging weights, sums up the results and adds a bias. Finally, it takes the result of this calculation as input for the activation function. The result of this is the output y. [38]

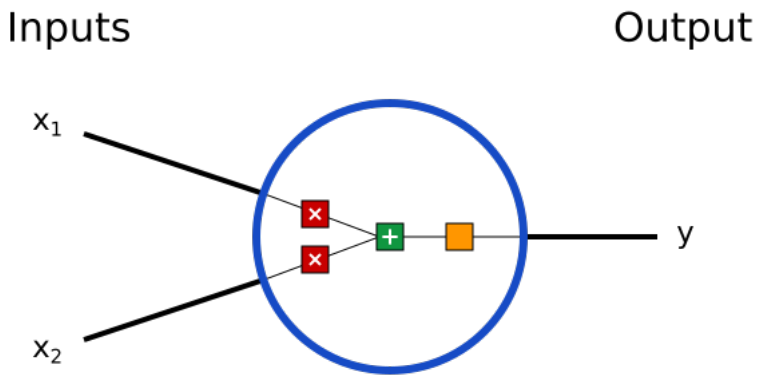


Figure 2.8: 2 input Neuron[38]

A neural network can consist of more than one hidden layer. In that case, these networks are called deep neural network. In figure 2.9 such a deep neural network with three hidden layers and two output values can be seen. [37]

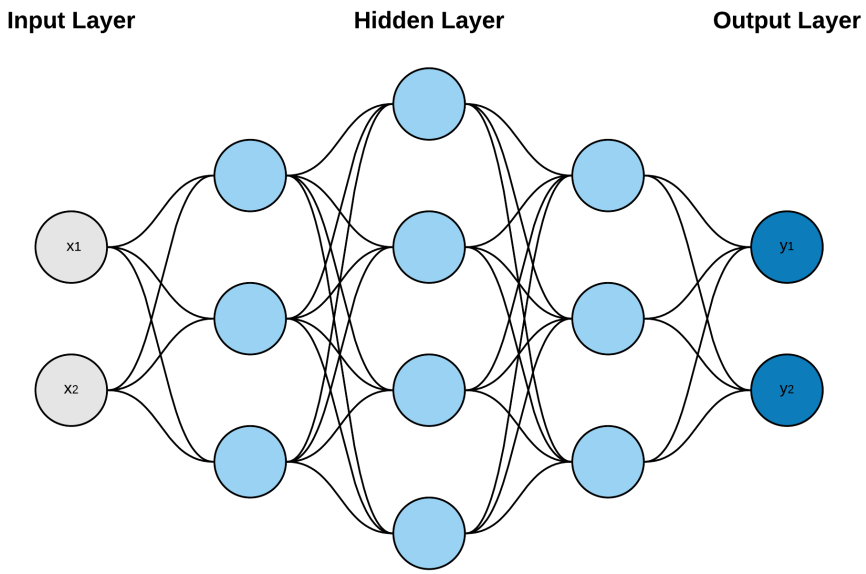


Figure 2.9: Deep neural network with 3 hidden layers

When training a network, the objective is minimizing the loss function. The loss function is a function, which provides information about how good the neural network is. One common loss function is the Mean Squared Error: [38]

$$MSE = \frac{1}{n}\sum *i = 1^{n}(y * true - y_{pred})^2 \tag{2.7}$$

With n as the number of samples, y_{true} as the true value of the variable and y_{pred} as the predicted value of it. To minimize this loss, the neural network uses the partial derivative to change the weights and biases accordingly. This is done over and over again for every sample until the network is well-trained with the given inputs.

Still, the developer does not only have to give some input data, train the neural network, and think all his work is done. Instead, other challenges have to be faced like preparing the data, choosing the right parameters and amount of input data to get the best results and avoid problems like over- or underfitting the network. How the developer handles these difficulties and what the necessary steps are in this new kind of development will be described in chapter 2.4. [35]

2.4 Artificial Intelligence lifecycle

The new possibilities opened by Machine Learning, as described in chapter 2.3 force developers to change their development lifecycle in a drastic way if they want to develop a Machine Learning based Artificial Intelligence. This lifecycle will be described and explained in the following chapter.

Important to mention is, that instead of code the developers have to produce for a specified objective, in Machine Learning the developers get some data as input and must train a model with this data to meet the objective. To simplify the process described in this chapter an example data set will be given and in the process of explaining the single steps this data will be used to exemplify these. This dataset can be seen below.

square meters	year built	year bought	age of buyer	type	# of rooms	price
50	2000	2005	33	Appartm.	3	250.000€
20		2010	26	Appartm.	1	100.000€
160	2004	2004	38	House	5	500.000€
300	2010	2016	41	House	7	680.000€
80	2018	2018	29	Appartm.	3	290.000€
48	1999	2008	24	Appartm.		220.000€

Table 2.1: Example dataset to predict the price of real estate

With this data the developer has to go through several steps to build a useful system based on them. These steps are defined in an open standard process model called Cross Industry Standard Process for Data Mining or CRISP-DM. This model can be seen in figure 2.10.

Following this standard model, the first step is Business Understanding. During this step, the project team has to determine overall goals and tasks for what to do with these data. For that, the team has to access the situation, which means that the team has to include possible risks, costs, benefits, and requirements for every possible objective in their evaluation process. After the overall objective is defined, this can be partitioned into several data mining goals what to do specifically with this data. Also, success criteria should be defined, and a project plan should be established, which will be followed in the next steps. [39]

Figure 2.10: CRISP-DM standard[40]

For the example above, such an objective could be to predict the price of real estate by the given features. The criteria could be, that in 90% of the tested data after the creation of the model, the predicted price should lie in between a range of 10% of the original price.

Next, the data need to be understood. For that, first, as much data as possible should be collected and analyzed to get a first insight. This data should then be evaluated by its quality to estimate the necessary effort to prepare and clean the data for building a model. Also, possible problems should be detected early to fix them as soon as possible. [39]

In the example, there are far too little data for a real model, and much more would be needed in a real project. However, even in this small dataset, some problems can be detected. The first one is the missing entry for the second object in the column “year built” as well as for the last object in the column “# of rooms”. Also, the type of estate is categorical instead of numerical, which could cause problems when training the model.

The next step is probably the one that needs the most effort of the developer - data preparation. First, the project team needs to select the data, which it will use for building the model. Then

the data need to be preprocessed. [39]

This preprocessing step includes detecting and removing noisy and redundant data. Noisy data means data that are irrelevant to the chosen business objective. In the given example, the age of the buyer is unnecessary information because it does not change the price of the estate. This circumstance is why this column can be removed. Also, wrongly labeled data should be removed.

Additionally, in the preprocessing step, unbalanced datasets should be balanced. This means that if one specific class of data is overrepresented and another one underrepresented, the dataset should be balanced so that the distribution of the different data classes are representative. In the example, apartments could be overrepresented, because there are twice as many apartments in the selected data than houses. The solution would be to add some more houses in the dataset or remove some apartments.

Last missing values have to be handled because null values could negatively influence the model. A better way would be to fill the missing cells with average values for this type of data. However, to choose the average value is not always the right way - it could be better to choose the minimum, for example. To decide the method of handling missing values is one challenging task for the developer. [41]

After the preprocessing, the dataset could be improved by feature engineering. The objective of this task is to improve the features such that the model better understand the coherences and improves its production function. [41] For example, new features could be created based on the knowledge of the data. A condition for that is to have an excellent understanding of the data. In the example above the developers could combine the column “year built” and “year bought” to “age at date of purchase” with a simple subtraction, which could give an essential insight of the price. Another meaningful feature could be some relation between the size of the real estate and the number of rooms.

Additionally, there can be features in a dataset that are not numerical but categorical. This data need to be encoded before training the model. The encoding technique depends on the context of the categorical data. One simple technique is label encoding, with which for every different category tha column gets a different number. For example every “Appartment” gets a

1 and every “House” a 2 in the “type” column. However, this could cause trouble, because the learning algorithm could interprete the numbers as sequence. That’s why another technique is One-Hot-Encoding. In this technique every possible category gets an own column and for every entry in the dataset the value is set to either 1 or 0 for all of the columns. In the example this would mean, that the column “type” would be replaced with the columns “appartment” and “house”. For every appartment the column “appartment” would then be set to 1 and the column “house” to 0. For every house it would be exactly the other way round.[42]

In the next step all collected and cleaned data should be integrated and merged. After that the feature values should be normalized, because there is usually a significant difference between the minimum and the maximum value of a feature. To increase the performance of a model it could be helpful to scale the values down to, for example, a range from 0 to 1. [41] One approach for this is called MinMax and converts the numbers through the following equation [43]:

$$z = \frac{x - \min(x)}{[\max(x) - \min(x)]} \tag{2.8}$$

In the end the example above could look like this:

square meters	year built	year bought	age at purchause	type	# of rooms	price
0.11	0.05	0.07	0.56	0.00	0.33	0.26
0.00	0.50	0.43	0.22	0.00	0.00	0.00
0.50	0.26	0.00	0.00	1.00	0.67	0.69
1.00	0.58	0.86	0.67	1.00	1.00	1.00
0.21	1.00	1.00	0.00	0.00	0.33	0.33
0.10	0.00	0.29	1.00	0.00	0.50	0.21

Table 2.2: Example dataset to predict the price of real estate

This data set has then to be splitted into three different sets - training, validation and testing. This is done to ensure that the model does not overfit to the training data. The model will be trained with the training set. Then the hyperparameters, which are used to improve the model with some different parameters dependent on the used model, are tuned with the help of the validation set. The test set is used to test the models actual performance at the end.

After this, the next step is modeling. For this, first, a training technique, as well as a basic

model, has to be selected. Usually, different models are tested several times, so that the best model can be chosen in the end. This model will then be used to train it with the training set. As already mentioned above the hyperparameters are then tuned with the help of the validation set. This prevents, for example, over- and underfitting the model with the training data. [39]

Then the model has to be evaluated with the test set. Also, the whole process needs to be reviewed and improved if possible. Dependent on the resulting model and the satisfaction of all stakeholders, the steps can be repeated starting from the Business Understanding with deeper knowledge. This is how a model can be continuously improved until all stakeholders are satisfied by tuning the hyperparameters, choosing different models or improved data and features, or applying different training methods. [39]

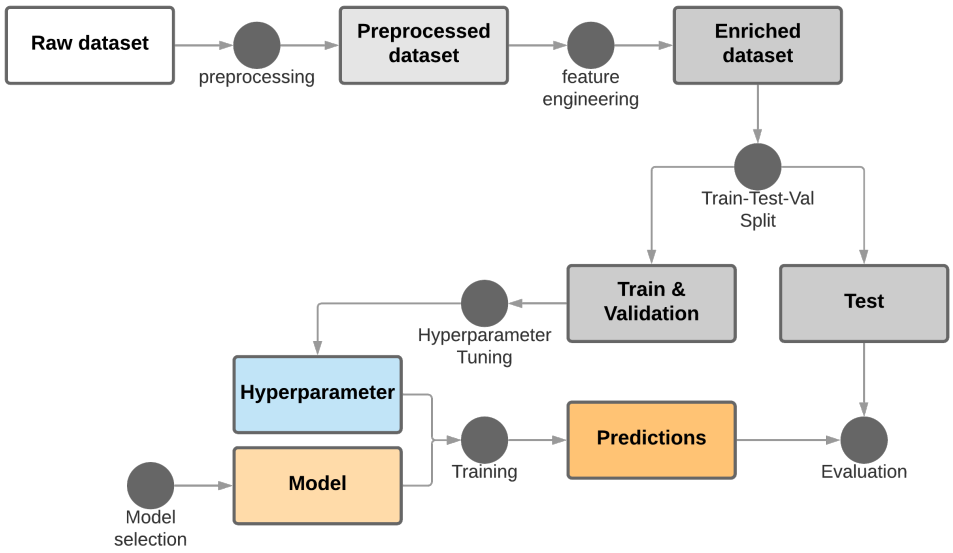


Figure 2.11: Iterative Machine Learning lifecycle

This process from preparing the data and adjusting the parameters can be seen in figure 2.11 from a more technical perspective. First, the developer gets the raw dataset, then he preprocesses this dataset and applies feature engineering to improve it. After it, the dataset is split into different sets, and the parameters get tuned. At the same time, a model is getting selected and trained with the training set and the given parameters. This step is repeatable until the developer is satisfied with the created model. In the end, the model's predictions

will be evaluated against the test data.

Last, the model needs to be deployed. For that also, the monitoring and maintenance of the product need to be clarified. This is how the model can be applied to its business case. The objective is an easy and stable way to access the new product. The cycle will then be finalized with a final report and review of the product as well as the process. [39]

With this standardized process of Artificial Intelligence development, the basics for applying DevOps to them are already existing. In chapter 2.5 will be described how these steps can be automated while simplifying the work for the developer and increasing the efficiency at the same time based on the principles and practices of DevOps described in chapter 2.1.

2.5 DevOps for Artificial Intelligence

The new steps described in chapter 2.4 forces developers to change not only their development cycle but also the operation. Additionally, new architectural models like microservices, cloud technologies like containerization or Kubernetes as well as SaaS as a new Software model as described in chapter 2.2 opens new possibilities of a scalable, flexible and reliable deployment of products. All this changes the way of DevOps for Machine Learning / AI, and not all of the existing principles and practices are applicable any more or better approaches are imaginable. Based on the common set of practices and principles of DevOps described in chapter 2.1 in this chapter these principles will be adopted and extended for applying it to AI development with the help of the new technologies presented in chapter 2.2.

First, DevOps for Machine Learning has to follow the principles named in chapter 2.1. To repeat, these are the following:

- Develop and test against production-like systems
- Deploy with repeatable processes
- Amplify a feedback loop [16]

Additionally, the four stages - steer, develop, deploy, and operate - also apply for Machine Learning. To adopt existing principles, these stages will be passed through, and necessary

adoptions or additions will be made.

The first set - **steer** - was about management and planning, which includes Continuous Business Planning, Continuous Improvement, and Release Planning. All this includes: Tracking the status and the needs of a project, monitoring a product and getting feedback from the users as well as tracking the progress of the project to minimize the risks and be able to react on trends as quickly as possible.

All this also applies to the development of AI and overlaps with the CRISP-DM process described in chapter 2.4. This process defines *Business Understanding* as the first stage, in which business goals and objectives should be defined. During this step, the same practices and tools can help as in traditional software development as there is no difference in planning and managing the objectives and release plan of a Machine Learning or a traditional software project.

The difference in the steer phase is that there is an additional step, which includes understanding the data. This understanding needs a deeper insight into the business correlations, necessary information, and context of the data. So, there are two significant differences -

The first is in the roles of the people who handle these steps. This profound insight of the business data needs experts on this field instead of programmers with a lack of understanding all the correlations and meanings. Instead, usually, data scientists or data analysts are needed for defining the needed data, evaluating the quality and detecting problems, so that a clear way to clean and prepare the data can be determined.ro

A second difference is a need for a tool to visualize these data. For that, the data scientists or analysts need skills for creating visualized data as quickly as possible. A widespread tool for this is *Jupyter Notebooks*, with which an easy preparation and plotting of the data are possible with the help of Python.

However, most differences are in the **development and testing** stage. While in traditional software development this step consists mainly of coding and testing the single components with unit tests as well as integration tests, in ML development this stage is split into two stages - data preparation and the building of the model. Also, the code is not the only input the developers have to deal with, but there is data as a second, valuable input. [9] This leads to

several differences in the practices and tools defined for this stage.

Starting with the *Collaborative development* and *Continuous integration*, the main point is to integrate and share not only the code between all participators but also the data. Usual source control software like git sets limitations to the file size and are not designed for handling other data than code. This results in need of another tool to handle big data samples when developing an AI product so that developers can share and integrate all of their work and not only the code. This is also necessary to share the results with the client and keep the product reproducible.

A solution to this problem could be Quilt or Git Large File Storage (LFS), which are designed for storing audio samples, graphics, datasets, and videos.

Another common tool for developing software are IDEs, which are supporting the developer in coding, collaborating with one's team, and efficiently integrating the workflow. For Machine Learning, such tools have to be extended, or new tools need to be created to visualize and especially prepare data. This includes, for example, possibilities to label images or videos and preprocess the data. Currently, for this, different tools have to be used. As long as there is no standardized way for labeling data, teams should agree on one tool for guaranteeing the correctness and uniformity.

However, tools for labeling data, managing its quality and collaborating with a team are emerging. An example is Labelbox, to name just one, which offers simple data labeling and management, collaboration, and even some automation features. This labeled and prepared data then needs to be available from a shared repository, to which the whole team can contribute to so that they can collaborate.

Another practice in traditional Software development is called *Continuous testing*. This practice includes automatically build the software and run unit tests on every single component as well as automated integration tests of the application as a whole. In AI development, a completely different form of testing is necessary:

First, a Machine Learning model can only be tested as a whole instead of its single components. Machine Learning models work more like a black box because it is difficult to look at how the

inner components are working and evaluate its actions. This means that only an input can be given and the output can be classified as true or false without a valuation of the single components.

Additionally, if the output is correct for one input that does not give any information about another input, because usually, ML models are too complex to predict its outcome. Both of it leads to another form of testing.

For this approach, the data needs to be split automatically into a training and a test set before training the model. After the training step is done, the test set is used to compare the predicted outcome with the real value. Applying this approach can give valuable insights into the model, which can be expressed as different indicators.

The easiest one is the accuracy, which divides the correct classified results by all tested samples:

$$Accuracy := \frac{\text{number of correct predictions}}{\text{number of all predictions}} \tag{2.9}$$

Another indicator is a confusion matrix. A confusion matrix shows the number of *True Positives* and *True Negatives*, which states correctly predictes positive or negative values, as well as *False Positives* and *False Negatives*, which states falsely predicted values. An example can be seen below:

n=165	**Predicted Positive**	**Predicted Negative**
Actual Negative	50	10
Actual Positive	5	100

Table 2.3: Example confusion matrix

[44]

A third indicator is the F1 score, which first calculates the precision as well as the recall of a model. Precision is the number of correctly predicted positive results divided by the number of predicted positive results. Recall on the other hand is the number of correctly predicted positive results divided by the number of all samples. The F1 scores combines those two

values to find a balance between them. The greater the F1 score, the better the performance of the model:

$$\begin{aligned} Precision &= \frac{\text{TruePositives}}{\text{TruePositives} + \text{FalsePositives}} \\ Recall &= \frac{\text{TruePositives}}{\text{TruePositives} + \text{FalseNegatives}} \\ F1 &= 2 * \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}} \end{aligned} \tag{2.10}$$

Last, also the mean squared error can be used as a measure for the performance of a model. The mean squared error calculates the average of the differences between the real value and the predicted value:

$$MSE := \frac{1}{N} * \sum *j = 1^{N} (y_j - \hat{y} * j)^2 \tag{2.11}$$

However, there are still several other indicators that can be used for measuring the performance of a model. The developer should decide which indicators to use, so that they will be seen to evaluate the performance of a model after the automated tests have been driven.

During the modeling stage, another important difference occurs: The urgent need for resources, especially of computing units and memory. In opposite to traditional Software development, this use does not equal the need for resources while running the application in production, but it needs far more resources to train the model. Usually, GPUs are used to build models because their design fits the need for model training better. However, GPUs are not very common in traditional Software development, so teams would have to purchase one just for this purpose. Additionally, these resources are only needed during the modeling stage, and the rest of the time, they would be unused, which is a waste of resources. This is where the advantages of Cloud Computing can be utilized. With its capability of flexible resource allocation, the resources can be used more efficiently, and money can be saved because of the pay-per-use model of Cloud Computing. This approach of saving cost and resources through training the model within a Cloud can be called *Dynamical resource demand*.

The next stage is the **delivery**, in which the practice *Continuous delivery* applies. This practice

deals with the automation of the deployment of the software to different environments. This demand also applies to Machine Learning development. Only the implementation differs slightly. In common Software development, the trigger for starting the building and deployment process is usually a newly committed code in the master repository. In Machine Learning, it could also be necessary or recommended to rebuilt and deploy the model when new or changed data occurs, which forces a second trigger. Additionally, the built of the project requires more steps than just compiling the code like training the model, which takes some time. This is described in more detail below when it comes to delivery pipelines.

An essential practice during the delivery stage to guarantee the principle of developing and testing against production-like systems is to containerize the ML applications and its contents. This guarantees a consistent environment during all stages, because the containers provide their environments on their own. The containers can then be deployed in any system, whether local or Cloud and the results should stay the same. This is already pretty common for traditional software but is often avoided with Machine Learning applications because of missing Cloud computing options. This is why Cloud providers are forced to build Cloud platforms, that enable the use of GPUs on a cluster so that containerized ML applications can become a reality. This would speed up the development of ML applications as well as it would increase its efficiency and reduce costs due to more comfortable ways of building, testing, and deploying these applications.

Last, there is the **operation** stage. During this stage, the use of the application should be monitored, and feedback should be collected. That information should then be used to improve future products and support the current version with updates and bugfixes.

For ML applications, this monitoring and feedback get one more, significant role - through the collection of more data, the model can be continuously improved through *Refitting of the model*. This is an exclusive feature for ML applications because, in the opposite of traditional software, these are adaptive. This demands a clean collection of the data as well as automated processing of these. Then this data can be used to retrain the model and redeploy it after periodically. This offers the users to experience a significant improvement of the app's performance, and that way increases the User experience due to better results.

Another critical difference is the roles of developing people. While in traditional development, the developers are IT specialists, who can handle the operational cycle as well as the development. In ML development, usually, Data Scientists take over the main part of the work as already mentioned above. They got a unique skill set when it comes to the preparation of data and feature engineering, but sometimes they lack in experience with operation tools like delivery pipelines. That is why a more natural way to build and operate a pipeline is necessary, which can be easily handled and reused so that the process of development is as easy as possible and the Data Scientist can focus on his main work with the data.

An essential part of a solution is to provide an easy-to-use delivery pipeline for the Data Scientists, which combines every step of the development and automates all the operations as far as possible so that the Data Scientist can focus on his main work.

An example draft of such a pipeline can be seen in figure {fig:devopsaipipeline}.

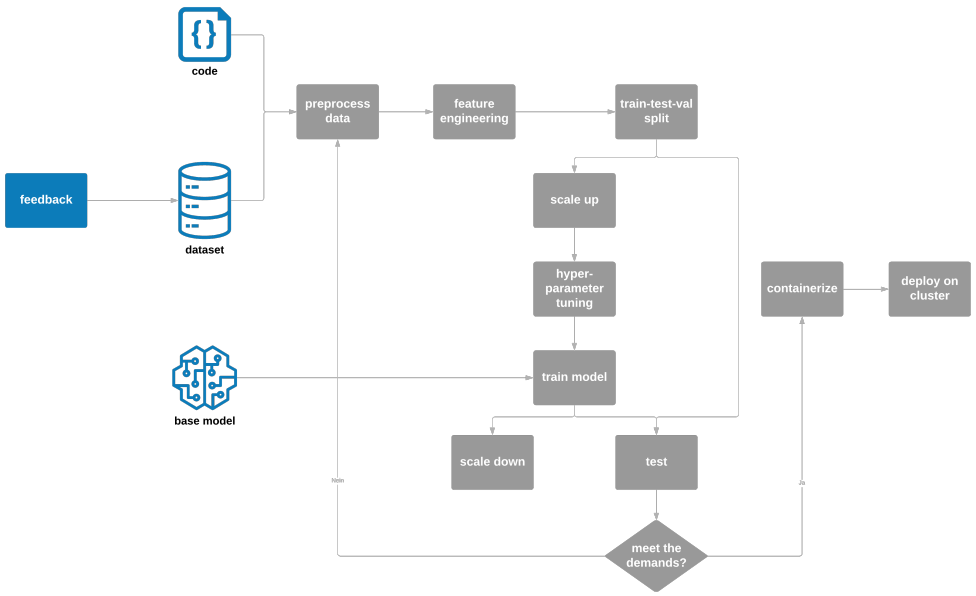


Figure 2.12: Example devops pipeline

There the first difference to traditional DevOps pipelines is, that there is another input than only the code. This is mainly the data, but also a base model, on which the model will be trained on. Both of them can optionally serve as a trigger for starting the delivery process. Alternatively, the process can be sequentially executed or bumped by the developers by

purpose.

Then there are a preprocessing as well as a feature engineering step. These two steps are realized by some scripts of the developers, which take over the preprocessing and feature engineering automatically. The cleaning and every other necessary step to be made with the data has to be done manually before pushing the data to its repository.

After this, the dataset is split into three sets - training, validation, and test. The training and validation sets are then used for the training of the model, while the test set is only used to test the finished model at the end.

However, before the training of the model, the pipelines scales up the cluster, so that enough resources for the training step are available. Also, the hyperparameter tuning can be done by automated scripts here. Alternatively, those hyperparameters can be given by input parameters before the pipeline is executed.

For the training of the model, which takes the main part of the time, the pipeline usually uses a given basic model, on which the model will be based on. This is done because building a completely new model from scratch needs much time and huge amounts of data, which is the reason why most models are based on some pretrained basic models and only some are built from scratch.

After this is being done, the cluster can scale down again, and the model can be tested with the test set. If the results are satisfying, the model and its application can get containerized automatically and then deployed on the cluster. The developer can define the way of how the application should be published and accessible from the cluster for the end-users. In case the model does not meet the requirements the data needs or the code needs to be adjusted or cleaned, or different hyperparameters should be chosen, so that a better model will be built in the next try.

Last it is to mention, that through feedback given by the users during the operation stage, the dataset is continuously extended by new data. This is why the pipeline should repeat the model building sequentially so that these new data can be used to improve the application continuously.

With these adjusted and complemented practices above as well as with a delivery pipeline similar to the described one, the principles of DevOps can be kept. Through the automated containerization and deployment on a cluster, every environment can look exactly like the production environment. If the whole pipeline and application should be tested first locally before deploying it on a Cloud, it can be tested on a local Kubernetes cluster like Minikube, which will be described in chapter 3.4.2.

Also, the delivery pipeline guarantees a repeatable and reliable deployment, in which almost every step can be automated, and the Data Scientist can focus on his work with the data. The only thing the developers need to do is providing scripts for an automated preprocessing and feature engineering if necessary.

The monitor and feedback principle has an even higher importance because through this, the application can be continuously improved through feeding the system with the newly gained knowledge.

This is why such a pipeline has significant importance in DevOps for AI. In the next chapters, two different approaches to building such a pipeline will be presented and evaluated with previously defined criteria.

3 Method - State of the Art

In chapter 2.5 DevOps practices have been adopted to the new world of AI development. One essential technology to meet those requirements is a delivery pipeline, which accompanies the entire development and deployment process. In this chapter, first, criteria will be defined to measure the success of a pipeline. Then some exemplary frameworks to create such a pipeline and other basic frameworks necessary for the development process will be introduced before it will be explained how to prepare the necessary environment for such frameworks and how to use them. Last, an exemplary problem will be created, whose solution will be presented in chapter 4.

3.1 Catalogue of criteria

For defining the criteria to measure the success of a delivery pipeline framework for AI development, this chapter will be oriented on the adopted and extended practices described in chapter 2.5.

First, it is important to mention that people who are dealing with these enormous amounts of data are usually other people with different skills than those who are usually dealing with IT operations. These Data Scientists are not mandatory skilled or even experienced with the delivery and operation of Software. This is why a framework for building such a pipeline should be easy to use and standardized as far as possible. The Data Scientists should have an easy time to set up the development environment, get familiar to the framework, and learn about the companies culture and working methods. This is why the first criteria to evaluate the different approaches is *simplicity*.

Besides an easy way to use such a delivery pipeline tool, it is also necessary that it is adaptable to other frameworks and technologies. This should be one of the main objectives, because technologies are changing quickly, especially in the field of AI. What is the most common tool nowadays can be completely outdated in a few weeks when an innovation breaks through. This is why tools for handling these technologies should be as *adaptable as possible* so that only small adjustments are necessary to change the basic framework or similar.

Another essential point for the developers that is different from traditional software development is the missing *IDE integration*. Because of some necessary scripts to handle the data during the delivery process, an integrated development environment would be helpful to support the Data Scientist in integrating the script within the delivery pipeline. An example of that would be an integration of Jupyter Notebooks.

Probably the most important point is the *support of a collaboration platform* for the data so that an entire team can work on them at the same time as it is already common with code. Additional *versioning* of the data as well as the resulting models would be welcomed as well.

Additionally, it would be necessary to *pass on some parameters* when starting the delivery process, for example, the base model or values for the hyperparameters.

To enable *scalability* to the delivery process, especially for the training step, the support of a cluster like Kubernetes would be needed. For this, every operation has to be containerized, so that it can run on the cluster. Ideally, the execution on Kubernetes or a similar cluster is directly integrated.

For a proper evaluation of the resulting model, good *visualization of the testing results* is desirable. The developer has to be able to see the results and react accordingly.

Last, the quality of the resulting models and the scope of functionalities supported by the frameworks is another important factor to measure.

With those criteria, the success of a framework and an accordingly delivery pipeline will be measured at the end of this work. If every requirement is met, it can be considered a successful framework for applying DevOps principles to the development and delivery process of AI.

3.2 Used frameworks and libraries

In this chapter, the used frameworks for creating such a pipeline will be introduced. Also Tensorflow, a Machine Learning framework, will be described shortly, with which the resulting models will be trained. Then it will be described how to create the necessary environment for using these frameworks, locally as well as on a Cloud.
Last, a concrete use case will be described, which will be implemented in chapter 4.

3.2.1 Tensorflow

To perform modern Deep Learning operations, there is no need to program a complete Neural Network on his own. There are several frameworks available; many of them are open-sourced. One of the most popular ones is Tensorflow.

Developed by Google as the successor of DistBelief, it had been open-sourced in 2015. Tensorflow is a large scale Machine Learning framework. Compared to DistBelief, the first generation of Googles ML frameworks, it offers a more flexible platform with better performance, which also supports a broader range of models and heterogeneous hardware platforms, so that it supports mobile, GPUs or clusters among others. This makes Tensorflow to one system for all scales, which leads to significantly fewer maintenance burdens in the operation cycle.
Additionally, TensorFlow achieves parallelism through replication and parallel execution, which raises the performance significantly. [45]

TensorFlow supports a very diverse range of tasks, starting with the inference of computer vision models on mobile phones to large-scale training of deep neural networks with hundreds of billions of parameters on hundreds of billions example records using many hundreds of machines. [45]

For the model building, Tensorflow offers the possibility to use high-level APIs like Keras. Keras is written in Python and supports different Machine Learning frameworks like Tensorflow, CNTK (Microsoft Cognitive Toolkit), or Theano. It works on CPUs as well as on GPUs and offers an easy and fast prototyping. [46]

For building a model, Tensorflow uses a programming model with directed graphs composed of a set of nodes. A graph represents a dataflow computation. Each node of a graph has zero or more inputs as well as zero or more outputs. It represents the instantiation of an operation. [45]
The values, that flow along regular edges in the graph, are called tensors. This can be compared to weights of the Neural Networks explained in chapter 2.3.3. [45]

To create a model, TensorFlow offers APIs for several programming languages like Python, Java, C++, Swift, JavaScript, or Golang. [47] To use this API, first, TensorFlow needs to be imported. Also, the dataset has to be prepared and loaded into the application. Assume this dataset is stored into the variable `data`, the dataset can be split into training and validation set as can be seen below:

```
1 (x_train, y_train), (x_test, y_test) = dist.load_data()
```

The epithet `x` datasets are for the input, the `y` datasets for the labels. After that, Keras can be used to build a model. For this, first, some layers need to be added. In this example, the first layer is the input layer, which receives a matrix in the size of 28x28. The second layer is the hidden layer. This layer is being initialized with 128 units and the sigmoid activation function. Last the output layer is defined, which gives ten outputs - one match probability for every category.

```
1 model = tf.keras.Sequential([
2     tf.keras.layers.Flatten(input_shape=(28, 28)),
3     tf.keras.layers.Dense(128, activation='sigmoid'),
4     tf.keras.layers.Dense(10, activation='softmax')
5 ])
```

After that the model has to be compiled with appropriate optimization, loss and metrics functions:

```
1 model.compile(optimizer=tf.keras.optimizers.RMSprop(),
2                 loss=tf.keras.losses.
```

```
                    SparseCategoricalCrossentropy(),
3                metrics=[tf.keras.metrics.
                    SparseCategoricalAccuracy()])
```

Last, this model can then be trained with some data. The num of epochs represents the number of how many times the model will be fed with the data :

```
1 model.fit(x_train, y_train, epochs=5)
```

This model can then be evaluated by comparing the predictions to the test set:

```
1 model.evaluate(x_test, y_test)
```

The evaluation function will compare the labels to the predictions and outputs the loss and accuracy of the model.

This development process enables the user to quickly build and train neural networks and made Tensorflow to one of the most used Machine Learning frameworks.

3.2.2 Kubeflow

As mentioned in chapter 2.4 Machine Learning development consists of more than just building a model. It also includes preparing and cleaning data, testing, scaling, and deploying the model. Automation of those steps would lead to significant performance growth.

However, the different steps of Machine Learning often require entirely different environments, which complicate the portability. Additionally, the used hardware differs a lot from step to step, which is the reason why scalability is an eminent point of efficient ML. All this makes the automation process much harder.

These problems can be solved by using Kubernetes. The reliability and performance that Kubernetes offers fit Machine Learning development perfectly, so the community started to utilize these benefits.

In doing so there are several challenges that need to be faced while trying to bring Machine

Learning applications to Kubernetes - the creation of the deployments as well as the managing of the services introduced huge barriers of complexity for data scientists, whose main work should be working with the data instead of caring about the delivery process. [48]

For facing these challenged Kubeflow has been created in 2017. Kubeflow is an open-source platform for Machine Learning to enable composable, postable and scalable ML on Kubernetes. The features offered by Kubeflow include a Jupyter Hub for collaborative and interactive training, a TensorFlow training resource and serving deployment as well as delivery pipelines with Argo Workflows. [48]

Argo Workflows is an open-source container-native workflow engine to orchestrate parallel jobs on Kubernetes. With Argo, workflows can be defined. Each step of such a workflow is designed as a container. This way Argo can run CI/CD pipelines on Kubernetes. [49]

With the help of Argo Workflow, Kubeflows offers a pipeline, which enables managing and tracking experiments, jobs, and runs in a multi-step ML workflow. In such a pipeline, all of the components of an ML workflow are included. This also requires the definition of the input parameters and delivers an output for each component. One component is a set of a containerized code and performs one single step in the pipeline. [48]

For creating such a pipeline, every single component has to be created as a container, and then the pipeline can be defined with the help of the Python SDK. Such a pipeline can look as below:

```
1 def gcs_download_op(url):
2     return dsl.ContainerOp(
3         name='GCS - Download',
4         image='google/cloud-sdk:216.0.0',
5         command=['sh', '-c'],
6         arguments=['gsutil cat $0 | tee $1', url, '/tmp/results.
              txt'],
7         file_outputs={
8             'data': '/tmp/results.txt',
9         }
```

```
    )


def echo_op(text):
    return dsl.ContainerOp(
        name='echo',
        image='library/bash:4.4.23',
        command=['sh', '-c'],
        arguments=['echo "$0"', text],
    )


@dsl.pipeline(
    name='Exit Handler',
)
def download_and_print(url=''):

    exit_task = echo_op('exit!', is_exit_handler=True)

    with dsl.ExitHandler(exit_task):
        download_task = gcs_download_op(url)
        echo_task = echo_op(download_task.output)
```

There, first, the two components have been defined. Within these components, some parameters are defined. First, the input parameters need to be given. In this example, the first step is to download an image, so the `url` has to be given as an input parameter. Then the component needs to be defined with several parameters. Except for the name these are the used Docker image, commands and arguments to be executed and the output of this step. In this example, the given file is downloaded, and the component forwards the output file to the next step. In this step, the content of this fill will be printed.

Then the pipeline itself is defined. The input of this function represents all the values the user

has to enter as inputs for the pipeline. In this example, this is only the URL of the file to be downloaded. Then the different steps are defined, and the inputs are forwarded. The output is stored in the variable `task_name.output`, so that other components can access and use it.

Last, the pipeline needs to get compiled with the Python SDK. Then the pipeline is created and can be uploaded to and executed with Kubeflow. How this works will be explained in detail with the help of an example in chapter 4.2

3.2.3 Azure pipeline

An alternative to an open-source solution like Kubeflow are Cloud integrated solutions. Every Cloud provider is developing its own solution for this problem, which becomes more and more important through time. So is Microsoft providing a Cloud-based environment to develop, train, test, deploy, manage, and track ML models with its Azure Machine Learning service.

Azure Machine Learning service supports different ML frameworks like PY Torch, TensorFlow, and Scikit Learning for any kind of Machine Learning. To offer support over the whole ML development cycle, it offers the use of different tools like a visual interface for building experiments and deploying models, a jupyter notebook to write the code or a Visual Studio Code extension to integrate the service directly into an IDE. [50]

This extension offers the user to create a team workspace which enables the possibility of Collaborative Development. The experiment can then be linked to the Cloud, on which this can then be run, computed, and monitored. The resulting model can then be registered to the Workspace so that it is persisted and can be used by other team members. [51]

The Azure Machine Learning service also provides a Python SDK, with which the user can start training his model locally and then scale out to the Cloud. [50]

With building a pipeline with the Azure ML service, it allows the user to automate the end-to-end ML process in the Cloud and reuse components. Also, the user can use different compute resources in each step, which increases the efficiency and lowers the cost of ML tasks, because not every step in the ML workflow requires the same amount of resources. [52]

An example of how this service can be used will be described in chapter 4.1.

3.3 Project objective and conditions

In this work, two example pipelines will be built and compared. For this, a sample model will be built, and every step of the AI development lifecycle will be gone through. The objective is to build a pipeline fulfilling every target and principle described in chapter 2.5.

For this, a public dataset will be used, called Fashion-MNIST. It has been put together by Zalando Research and contains 60.000 training and 10.000 testing images of different garments with a size of 28x28 pixel. This dataset has been converted to a CSV file on kaggle.com [53]. This version has been used to manipulate the data for preparing a more realistic dataset with noises and faults, which have to be removed. This way, all the steps of AI development can be run through realistically. [54]

For this, first, the training and testing datasets have been read and merged:

```
train_data = pd.read_csv('data/fashion-mnist_train.csv')
test_data = pd.read_csv('data/fashion-mnist_test.csv')
test_data.dropna(axis=1)
data = train_data.append(test_data, ignore_index=True)
```

Then, the labels have been manipulated, so that the category labels are given as strings to show, what the pictures actually represent. This forces the developer to clean these data later during the preprocessing step.

```
class_names = {0 : 'top', 1 : 'trouser', 2: 'pullover',3 : '
    dress', 4: 'coat',
                5 : 'sandal', 6 : 'shirt', 7 : 'sneaker', 8 : '
                    bag', 9: 'ankle_boot'}

data.label = [class_names[item] for item in data.label]
```

Then, some faulty values have been inserted, which has to be removed later:

```
import random

manipulated_vals = ['tshirt', 'hat', 'jacket', 'accessoire', '
    facemask']

for i in data.index:
    rnd = random.random()
    if rnd <= 0.001:
        data.loc[i,'label'] = random.choice(manipulated_vals)
```

For that, it will be iterated through every row, and some randomly chosen rows will be manipulated in a way, so that value of the label of this row is changed to another category, which is not truly existing.

Last, also some missing values have been inserted so that these rows have to be deleted or manipulated during the development to avoid errors during the model building process:

```
import numpy as np

rand_zero_one_mask = np.random.randint(100000, size=data.shape)
data = data.where(rand_zero_one_mask!=0, "")
```

For that, a mask of a random matrix with the same size as the dataset is being created with an even distribution of numbers from 0 to 100.000. Then the dataset will be compared to this matrix, and every cell that faces a 0 on its position in the matrix will be changed to an empty value.

This dataset is then stored as a CSV and will be used for the example implementation, which will be described in chapter 4.

3.4 Creating the necessary environment

To use Cloud integrated solutions, the installation and access process can be done quickly by clicking through the Cloud console. However, the installation of an open-source tool like Kubeflow can be more complicated.

Another challenge can be to create a local Kubernetes cluster to test the development processes on a local system first.

How these installations can be done is described in this chapter.

3.4.1 Azure Machine Learning service

The Azure Machine Learning service is provided on the Microsoft Azure Cloud. For accessing the Azure Cloud, a confirmed Microsoft account is needed. Then the Cloud can be accessed via portal.azure.com.

First, a resource group has to be created. For that the *Resource groups* button has to be selected on the left sidebar. Then a new group can be added by clicking on *Add*. In the upcoming form, a name has to be given as well as a subscription model has to be chosen. For non-paying users, the only possible subscription model is the free model. Additionally, a region can be chosen.

Next, a resource has to be created with the *Create Resource* button on the left side panel. There a Machine Learning workspace has to be created. For that, it can either be navigated to the category *AI + Machine Learning* or directly be searched for *"Machine Learning*" via the search bar. Then the *Machine Learning service workspace* has to be selected.

On the next page, this workspace can then be created by clicking *Create*. Then a name has to be given and the region, as well as a subscription model, has to be chosen the same way as for creating a resource group. Additionally, the newly created resource group has to be selected.

This new resource can then be accessed via the *All resources* page, which can be found on the left side panel. There the service appears with the chosen name. When clicking on the service, a new page with a navigation panel is loaded. To create a visual interface for a Machine

Learning pipeline, the *Visual Interface* button has to be clicked, which can be found in the category *Authoring (Preview)*. Then this interface can be launched by pressing *Launch visual interface* on the appearing page.

After that, the project is successfully prepared for creating a pipeline with the Azure visual interface.

3.4.2 Minikube

A critical principle of DevOps is to design the environments in a way that they behave as similar to the production environment as possible. Because of the decision to deploy the product inside of containers on a Kubernetes cluster, it is necessary to set up such a cluster on a local system. Currently, there are two popular solutions for this - Minikube and microk8s.

Minikube can start a Kubernetes cluster on the local device either within a VM or without a VM directly on the system. Microk8s always start the cluster directly on the system.

The advantage of a cluster running on the VM is the isolated environment. Additionally, the available resources can be precisely defined by the developer so that the cluster can not request too many resources needed for the system itself. On the other hand, it can be more challenging to access the systems resources such as GPU, and it is more static in resource allocation.

An advantage of microk8s compared to Minikube is the easy possibility to configure the cluster for the use of GPU with a single command. In case of Minikube, the Docker environment has to be changed for doing so.

However, microk8s is still in development and unstable, which lead to some trouble with deploying Kubeflow on it during the implementation process of this work. This is the reason why Minikube has been used to realize the local creation of a cluster.

For doing so, first, Minikube has to be downloaded. This can be done on Linux Ubuntu by executing the following commands:

```
$ curl -Lo minikube https://storage.googleapis.com/minikube/
    releases/latest/minikube-linux-amd64
$ chmod +x minikube
$ sudo cp minikube /usr/local/bin && rm minikube
```

First, it downloads the latest Minikube version for Linux. Then it changes the execution privileges and moves the downloaded sources to the `usr/local/bin` directory.

Then a special version of Docker must be downloaded to support the use of nVidia GPUs. This version is called nvidia-docker2 and be installed like this:

```
$ curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | \
sudo apt-key add -
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/
    nvidia-docker.list | \
sudo tee /etc/apt/sources.list.d/nvidia-docker.list

$ sudo apt-get update
$ sudo apt-get install -y nvidia-docker2
```

First, the gpgkey is added to the Linux packaging tool. Then the packaging tool has to be updated before nvidia-docker can finally be installed.

After this, the default Docker runtime has to be changed in the file `/etc/docker/daemon.json`. In there the line

```
"default-runtime": "nvidia",
```

has to be added at the beginning of the file. Afterward, Docker needs to be restarted:

```
$ service docker restart
```

Last Minikube can be started with GPU support through executing following command:

```
$ minikube start --vm-driver=none --apiserver-ips=127.0.0.1 --
    apiserver-name localhost --cpus 4 --memory 4096 --disk-size
    =15g
```

The VM-driver is set to none, and the API server is set to localhost. Then the needed resources are defined. In this example, 4 CPU cores, 4GB of memory, and 15GB of disk space are sufficient.

With this setup, Minikube can run on a local system similar to a Kubernetes Cloud deployment, which enables the possibility to develop and test against a system, which looks very much like the production system.

3.4.3 Kubeflow

No matter if using a local Kubernetes cluster or one on Cloud Kubeflow needs to be deployed on the cluster. For the purposes described in chapter {sec:usecase} only the pipeline part of Kubeflow is needed, which is why in this chapter, only the installation of this part will be explained.

This installation can be done by applying the YAML manifest provided inside the Kubeflow Github repository to the Kubernetes cluster. For that, first, the wished version needs to be defined before it can get applied

```
$ export PIPELINE_VERSION=master
$ kubectl apply -f https://raw.githubusercontent.com/kubeflow/
    pipelines/$PIPELINE_VERSION/manifests/namespaced-install.yaml
```

Then the namespace is created, and every service and deployment gets installed and deployed. The status of the pod creation can be checked by executing

```
$ kubectl get pods -n kubeflow
```

As soon as every pod is up and running, the port of the pipeline UI (User Interface) needs to be forwarded:

```
1 $ kubectl port-forward -n kubeflow svc/ml-pipeline-ui 8080:80
```

Finally, the Kubeflow UI can be accessed by opening `localhost:8080` on the browser. Now pipelines can be uploaded, and experiments with them can be executed on Kubeflow.

4 Result - Pipeline Creation

In chapter 2.5, the basics of DevOps for ML have been discussed and explained. One crucial point of this is a pipeline, which supports the development of AI product during the whole development lifecycle. The goal is to automate the single steps and to build reusable components so that the focus of work can lay on the main work with the data. Different approaches are existing for such a pipeline. In this chapter, two of these will be explained and implemented. The first one will be a visual interface of the Machine Learning Service by Microsoft Azure. With this interface, the single components can be selected and linked together, so that the single steps to build an AI model can be automated with this pipeline. The second approach is called Kubeflow. Kubeflow is an open-source framework running directly on Kubernetes, with which components can be written in Python and linked together within a pipeline with the Kubeflow SDK (Software Development Kit) for Python.

4.1 Azure pipeline

With the Azure Machine Learning service, Microsoft offers different ways for automating Machine Learning. One of these is a visual interface for building the single development steps as components and connect them such, that an automated pipeline is enveloping.

The only requirement for this is an activated account on Microsoft Azure. Then the user can create a new Machine Learning Service workspace directly on the dashboard. From this workspace, the user can launch the visual interface for building a pipeline.

In this workspace, a bunch of different components is selectable. They are split into several categories:

- Saved Datasets for selecting the dataset to be used
- Data Format Conversions for convert the type of the dataset
- Data Input and Output for manually importing or exporting data
- Data Transformation for manipulating, sampling, splitting and scaling datasets
- Machine Learning for building, training and evaluating models
- Python/R Language Models for adding scripts that deal with the datasets or models
- Text Analytics for processing and preparing texts for Machine Learning
- Web Service for connecting web services

For the described objective above, first, the dataset created in 3.3 has to be imported. For that, this dataset has to be uploaded via the New Dataset dialog. Then this dataset can be selected from the category `Saved Datasets`/`My Datasets`. The component can then be dragged to the workspace.

First, the columns, with which the work will be done, has to be chosen. With the component `Select columns in Dataset`, this can be done quickly. The dataset output has to be connected to its input, and then the columns can be selected in the configs of this component. By clicking on a component, some configurations can be made. For this component, all the columns to be worked with can be selected as can be seen in **??**. The output will then be connected to the next step.

In this case, this is the cleaning of the data. For this, there is a corresponding component in the category `Data Transformation` / `Manipulation`. After dragging it to the workspace, it has to be chosen, which columns should be tested for missing data. Because in the used dataset, every single cell is required to have a value, every column is being selected. Then it has to be set, what should happen with faulty rows. There are six options that can be performed: Removing the entire row when a missing value occurs within this row; Removing the entire column when a missing value occurs within this column; Replacing it with the mean, median or mode value or replacing it with a user-specified value.

There are not many missing values within this huge dataset so that the removing rows influence the overall performance of the creating model noticeably. Because of this and for simplicity reasons, the method to remove rows with missing values has been chosen.

Next, the data need to be normalized, which means that the values of the single pixels, which lay between 0 and 255, should be transformed into a value between 0 and 1. This can be done with the `Normalize Data` component. There the *MinMax* transformation method has been chosen, which has been described in chapter **??**.

Other possibilities would have been Zscore, Logistic, LogNormal, or TanH.

After that, the data has been prepared. However, the features can be improved with Feature Engineering. One example is to encode the categories so that they are given as numbers instead of strings. For that, there is no specific component existing, so it can only be done by writing and adding a Python script. The Python Script component can get three inputs - two datasets and a script bundle as *zip* for providing necessary methods. The script to encode the data can look as follows:

```
1  import pandas as pd
2  from sklearn import preprocessing
3
4  def azureml_main(df = None, df2 = None):
5      label_encoder = preprocessing.LabelEncoder()
6
7      df['label']=label_encoder.fit_transform(df['label'])
8      df['label'].unique()
9
10     return df
```

The encoding is being done with the help of Scikit-Learn and assigns every category a number. Then the new data frame is returned and is used as output. A second data frame could have been returned as well and function as a second output of the component, but in this case, there is only one data frame necessary as input as well as output.

Now the data needs to be split to a training dataset and a testing dataset. The `Split Data` component provides this functionality and only needs to be connected to a data frame as input. Then the size of the training set can be configured, and it gives both datasets as output - training as well as testing.

After that, every necessary step has been completed to start the training of the model. However, for this, a model needs to be built before. Because this model-building needs no input, it can be done parallel to preparing the data. For the model building, there are several components with different models. For this image classification task, a `Multiclass Neural Network` is the component to use.

There, the number of hidden nodes, the learning rate, the number of learning iterations, and the momentum can be configured. For this example, 128 hidden layers have been used, and 100 iterations have been made. The learning rate, so the size of the step taken at each iteration, has been set to 0.1. The momentum, which is a weight to apply to nodes, has been set to 0 because there are no previous iterations to take this information from.

Then the `Train Model` node can be added to train the model. For this, it needs the model on the one hand and the dataset on the other hand as input. Additionally, the target column has to be chosen, which should be predicted by the model. In this dataset, this target column is the *label* column.

After the training is finally done, it can be evaluated. For that, first, it needs to get scored via the `Score Model` node with the model and the test data as input. The result of this can then be visualized with the `Evaluate Model` component.

The resulting pipeline then looks as can be seen in Appendix 6.1.

This can be run on an Azure cluster by clicking Run. After it has been completed, the result of the evaluation can be seen by right-clicking the output of the `Evaluate Model` component and selecting *Visualize*. This looks like below:

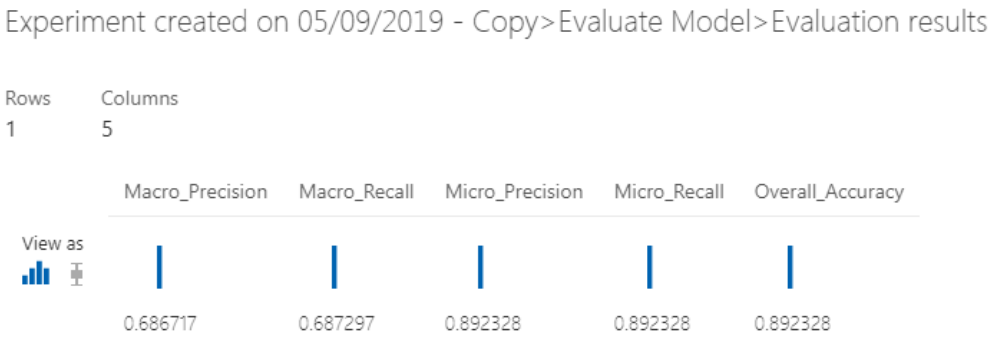


Figure 4.1: Microsoft Azure Machine Learning Service evaluation

As can be gathered from figure **??**, the accuracy of the resulting model is about 89%. If this is considered good enough, it can now be connected to a Web Service or similar. Otherwise, some parameters can be edited, or additional steps can be added to improve the model further. This makes this pipeline very easy to use to build models with existing data quickly.

4.2 Kubeflow pipeline

Another approach is to build a pipeline with an open-source framework like Kubeflow. Kubeflow provides a Python SDK for building pipelines and a visual interface running on Kubernetes for starting and controlling them.

The creation of a pipeline is broken down into two steps - the definition of the single components and the assembling of those to the pipeline.

For every component, there must be a Docker container, which performs the necessary calculations and actions. Data can be forwarded to these data by parsing arguments to it.

To explain the process of building pipelines, first, the single components will be shown and explained. Then the creation of the pipeline, including the definition of the components as well as the assembling, will be described.

First, the whole AI development process has to be split into several steps. Every single step needs its own container. For the described objective from chapter 3.3, the steps of AI develop-

ment described in chapter 2.5 will be applied. This means that the process can be split into the following steps:

- Data preparation
- Feature Engineering
- Data Split
- Model building
- Model Training
- Model evaluation

These will be investigated and described first before the creation of the pipeline itself will be explained.

4.2.1 Components

During the preparation steps the initial dataframe gets loaded and manipulated such, that there are no missing or faulty values and, that the data has been normalized. This has been realized with following Python script:

```
1  def main():
2      parser = argparse.ArgumentParser(description="Preprocessing"
           )
3      [...]
4      args = parser.parse_args()
5
6      df = load_data(args.dataset_location)
7
8      df = unify_datatypes(df)
9
10     df = remove_mis_values(df)
11
12     df = remove_faulty_values(df)
13
```

```
14      df = normalize_values(df)
15
16      df.to_csv(args.output, index=False)
17
18      write_file("/prepdf_output.txt", args.output)
```

There the main function gets started when the file is executed. This function first adds some input parameter. In this, there are two - the location of the original dataset, on which the transformations should be done, and the output location of where to store the resulting, cleaned dataset.

After that, first, the dataset is being loaded into the script. Then the datatype gets unified because some columns are recognized as Integer and some as Floats. For this, every column being recognized as Integer is converted to a Float and the appropriate field gets overwritten.

```
1  def load_data(path):
2      return pd.read_csv(path)
3
4  def unify_datatypes(dataset):
5      for col in dataset.select_dtypes("int64"):
6          dataset[col] = dataset[col].astype("float64")
7      return dataset
```

Next, rows with missing values are being removed. For this, every empty cell is being replaced with a missing value sign. Then all of them gets removed by the `dropna` function provided by the *pandas* library.

```
1  def remove_mis_values(dataset):
2      dataset.replace('', np.nan, inplace=True)
3      dataset = dataset.dropna()
4      return dataset
```

Also, rows with faulty values have to be removed. For this, the rows have to be removed, in

which the label does not equal one of the valid categories. These have been defined in the `CATEGORIES` array. This is done by only keeping the rows which label is included in this array::

```
CATEGORIES = ['top', 'trouser', 'pullover', 'dress', 'coat', '
    sandal', 'shirt', 'sneaker', 'bag', 'ankle_boot']

def remove_faulty_values(dataset):
    dataset = dataset[dataset.label.isin(CATEGORIES)]
    return dataset
```

The last step is to normalize the data. For this over every cell excluding the label cell is being iterrated and every value is divided by 255.0:

```
def normalize_values(dataset):
    for col in (col for col in dataset.columns if col not in ["
        label"]):
         dataset[col] = dataset[col] / 255.0
    return dataset
```

This can be done because every value lays between 0 and 255, so that the MinMax equation results in:

$$z = \frac{x - \min(x)}{[\max(x) - \min(x)]} = \frac{x - 0}{[255 - 0]} = \frac{x}{255} \tag{4.1}$$

Last, the dataframe is saved as *CSV* and the output location is stored to the given output file:

```
def write_file(path, content):
    f = open(path, "w")
    f.write(content)
    f.close()
```

This needs to be done because Kubeflow uses txt files to transfer the location of stored files to the next component.

The next component should deal with the feature engineering to optimize the features and labels for the training process. For example, in the described case it could be helpful to transform the label of the cateogory to a numerical value. The performance of the resulting model can be further improved by applying One-Hot-Encoding described in chapter 2.4. This can be done with following script:

```
1  def main():
2      parser = argparse.ArgumentParser(description="Feature
           engineering")
3      [...]
4      args = parser.parse_args()
5
6      df = load_data(args.dataset_location)
7
8      df = one_hot_encoding(df)
9
10     df.to_csv(args.output, index=False)
11
12     write_file("/findf_output.txt", args.output)
```

First, the arguments are added. These are once again the location of the dataset and the output location of the resulting dataset. Then the data gets loaded the same way as in the data processing component. Afterward, the one-hot encoding is being executed.

For this the *pandas* library can be used to grab the different categories, building an own column for each of them and filling the cells with either 1 or 0, dependent on whether the cell had equalled the according value before. Then the original label column gets dropped and the resulting columns of the described operations are joined to the dataset:

```
1  def one_hot_encoding(dataset):
2      one_hot = pd.get_dummies(dataset['label'])
3      dataset = dataset.drop('label',axis = 1)
4      dataset = dataset.join(one_hot)
```

```
5       return dataset
```

Afterward, the dataset is stored into a *CSV* file and its location to a *txt* file the same way as at the end of the data preparation component.

The last part of handling the data is to split it into different sets - training and testing. Following scripts takes over that task:

```
1  def main():
2      parser = argparse.ArgumentParser(description="Feature
           engineering")
3      [...]
4      args = parser.parse_args()
5
6      df = load_data(args.dataset_location)
7
8      image_df, label_df = split_label_and_img(df)
9
10     images_train, images_test, labels_train, labels_test =
           train_test_split(image_df, label_df, test_size=args.
           test_size, random_state=args.random_state)
11
12     images_train.to_csv(args.output_train_img, index=False)
13     labels_train.to_csv(args.output_train_label, index=False)
14     images_test.to_csv(args.output_test_img, index=False)
15     labels_test.to_csv(args.output_test_label, index=False)
16
17     write_file("/trainimg.txt", args.output_train_img)
18     write_file("/trainlabel.txt", args.output_train_label)
19     write_file("/testimg.txt", args.output_test_img)
20     write_file("/testlabel.txt", args.output_test_label)
```

In this script, first, the required arguments are defined. On the one hand, these are the

locations of the input as well as the resulting datasets. On the other hand, also the size of the testing set and a random state, with which different compositions of the datasets can be created, has to be given as a parameter.

Then the dataset is being loaded as described above, before the labels and the image informations are split. For this, two datasets are created - one, which consists of all columns, that includes the keyword *"pixel"*, and one of all columns, that does not include this keyword:

```
1 def split_label_and_img(df):
2     images = df.drop([col for col in df.columns if 'pixel' not
         in col ], axis='columns')
3     labels = df.drop([col for col in df.columns if 'pixel' in
         col ], axis='columns')
4
5     return images, labels
```

Next, the `train_test_split` function included in the *Scikit-Learn* library can be used to split those datasets into testing and training sets. These are then stored as four different files - the training images, the training labels, the testing images, and the testing labels. The output location is then written to a *txt* file once again.

Parallel to this data preparation, also the model has to be prepared. The layers has to be defined and the model needs to be compiled. Following script offers such functionalities:

```
1 def main():
2     parser = argparse.ArgumentParser(description="Feature
         engineering")
3     [...]
4     args = parser.parse_args()
5
6     model = build_model((args.input_shape_height, args.
         input_shape_width), args.num_units, args.num_outputs,
         get_activation_func(args.activation_l2),
         get_activation_func(args.activation_l3))
```

```
7
8       model.compile(optimizer=args.optimizer,
9           loss=args.loss,
10          metrics=[args.metrics])
11
12      model.save(args.output)
13
14      write_file("/model.txt", args.output)
```

First, several arguments are passed. These are the input shape of the images, the number of units in the hidden layer, the number of output categories, the activation functions for the different layers and optimizer, loss and metrics parameters for compiling the model. Then the model needs to be built. For this, Keras is being used to define the single layers and enters all the parameters:

```
1 def build_model(input_shape, num_units, num_outputs,
     activation_l2, activation_l3):
2       return keras.Sequential([
3           keras.layers.Flatten(input_shape=input_shape),
4           keras.layers.Dense(num_units, activation=activation_l2),
5           keras.layers.Dense(num_outputs, activation=activation_l3
                )
6       ])
```

To get the activation functions, the `get_activation_func` function gets a string of the functions name and returns the function itself. After that, the model needs to be compiled with the loss function, the optimizer, and the metrics to be used. This model is then saved and its location stored into a *txt* file.

Alternatively a prexisting and possibly pretrained model can be downloaded. One example for this can be found in the following script:

```
1 def main():
```

```
    parser = argparse.ArgumentParser(description="Feature
        engineering")
    [...]
    args = parser.parse_args()

    model = download_model((args.input_shape_height, args.
        input_shape_width))

    model.save(args.output)

    write_file("/model.txt", args.output)
```

There, only the input shape and the location of where the downloaded model should be stored has to be parsed as a parameter. Then the model can be downloaded with the given input shape and gets stored afterward.

No matter if the model has been built from scratch or a preexisting model has been downloaded, the next step is to train the model by feeding it with some data.

```
def main():
    parser = argparse.ArgumentParser(description="Feature
        engineering")
    [...]
    args = parser.parse_args()

    train_img_raw = load_data(args.input_train_img)
    train_label = load_data(args.input_train_label)

    train_img = prepare_image_shape(train_img_raw, args.
        input_shape_height, args.input_shape_width)

    model = load_model(args.model_location)
    model.fit(train_img, train_label, epochs=args.epochs)
```

```
13
14      model.save(args.output)
15
16      write_file("/trained_model.txt", args.output)
```

For doing this, the script needs the location of the image as well as the label dataset and of the basic as well as the resulting model. Additionally, it needs to know about the input shape for the images and the number of epochs, which means the number of iterations of feeding the model with data to complete for the training. Afterward, the data gets loaded, and the images get shaped correctly:

```
1 def prepare_image_shape(imageset, shape_height, shape_width):
2     return np.array([img.reshape(shape_height,shape_width) for
          img in imageset.values])
```

For this, the `reshape` method for arrays of the *numpy* library can be used. Then the model is loaded too, and it can be fitted with the loaded and prepared data. Last, this model gets stored, and its output location is written into a *txt* file.

The last step is then to evaluate this model. For this it needs the same parameters as for training the model with the only difference, that it gets the location of the testing datasets instead of the training datasets:

```
1 def main():
2     parser = argparse.ArgumentParser(description="Feature
          engineering")
3     [...]
4     args = parser.parse_args()
5
6     test_img = load_data(args.input_test_img)
7     test_label = load_data(args.input_test_label)
8
9     test_img = prepare_image_shape(test_img.values, args.
```

```
        input_shape_height, args.input_shape_width)

    model = load_model(args.model_location)

    loss, acc = model.evaluate(test_img, test_label)

    store_loss_acc(args.output, loss, acc)

    write_file("/result.txt", args.output)
```

Then all the data gets loaded, the images prepared and the model evaluated. During this operation, the accuracy, as well as the loss, are getting calculated. After the evaluation has been finished, these information are written to an artifact, so that it can be visualized in the pipeline:

```
def store_loss_acc(file, loss, acc):
    metadata = {
        'outputs' : [
        {
            'storage': 'inline',
            'source': '# Results\n Accuracy: {} \n Loss: {}'.
               format(acc, loss),
            'type': 'markdown',
        }]
    }
    with open (file,'w') as f:
        json.dump(metadata, f)
```

For this a JSON is being created, which stores the results in markdown format. This is then written to the output file, which can be used by the pipeline to visualize this markdown.

With all these components being created, these need to be packed into a Docker container. For this, a Dockerfile is necessary, which has to look similar to the one below:

```
FROM Python:3

COPY ./requirements.txt .

RUN pip install -r requirements.txt

COPY ./data_preparation.py .

ENTRYPOINT ["python", "/data_preparation.py"]
```

The base image of the Docker containers is a plain Python image. On top of it, the requirements are getting installed, which includes every necessary library and framework, and then the script itself is being loaded into the Docker container. This file then represents the entrypoint of the container with the possibility to pass additional arguments to it, so that every necessary information can be provided.

The Docker container of every component looks similar with the only difference that the components using Tensorflow are not built on a plain Python image, but on the latest Tensorflow image.

Then these Docker containers are build and finally pushed to a repository, from where they can be accessed by the pipeline.

4.2.2 Building pipeline

With every container build and pushed the creation of the pipeline itself can start. For this, first, several components of the Kubeflow Python SDK needs to be imported:

```
import kfp
from kfp import components
from kfp import dsl
from kfp import onprem
```

Then the single steps or operations to be executed during the development have to be defined as functions. Such a function looks like below:

```
def data_prep_op(dataset_location, output):
  return dsl.ContainerOp(
    name='Data preparation',
    image='pascalschroeder/kf-dataprep',
    arguments=[
      '--dataset_location', dataset_location,
      '--output', output
    ],
    file_outputs={
      'output': '/prepdf_output.txt'
    }
  )
```

The function gets the necessary parameters as input. Then it returns a ContainerOperation, defined in the Kubeflow Python SDK, with a given name, the docker image to be used and arguments as well as file outputs.

The arguments have to equal the arguments, that have been parsed during the creation of the components. By defining these arguments in the definition of the operation, these arguments can be given to the pipeline and forwarded directly to the component, that needs this information.

The file outputs represent the files being created, in which the information needed for the following steps are being stored.

Each of the components needs to have an operation defined looking like this or similar. Then these operations can be connected as can be seein in Appendix 6.2

This pipeline first needs to define every parameter that will be used for one of the components. It also declares a default value in case the user will not change the information on the startup of the pipeline.

Then the single operations are being called. By calling these operations, every needed information is being forwarded as a parameter. To use an output of a previous step as new input, the according parameter of the outputs array of the previous step can be called.

In the example case, both should be tested - a new model built from scratch as well as a pretrained one downloaded from the internet. Different paths can be defined for these conditions depending on the results obtained.

After every step is defined and called the pipeline can be compiled with the Kubeflow SDK like this:

```
if __name__ == '__main__':
  kfp.compiler.Compiler().compile(pipeline, __file__ + '.tar.gz'
     )
```

This only needs the file to be started usually and then created an archive, which can be imported to the Kubeflow dashboard.

After this have been done the created pipeline looks like can be seen in Appendix 6.3

The user can then start this pipeline. When starting it, the user can enter or change all the parameters defined above, as can be seen in Appendix 6.4.

Finally, the pipeline gets started, and every single step will be gone through automatically. With the manipulated *Fashion-MNIST* dataset created in chapter 3.3 the accuracy can be measured and lays at about 98% while a loss of about 5%. This result can be visualized with an artifact. This artifact file replaces the `file_outputs` field of the defined operation:

```
output_artifact_paths={
            'mlpipeline-ui-metadata': '/mlpipeline-ui-metadata.
               json',
    },
```

This output artifact then creates a visualization on the pipeline looking like below:

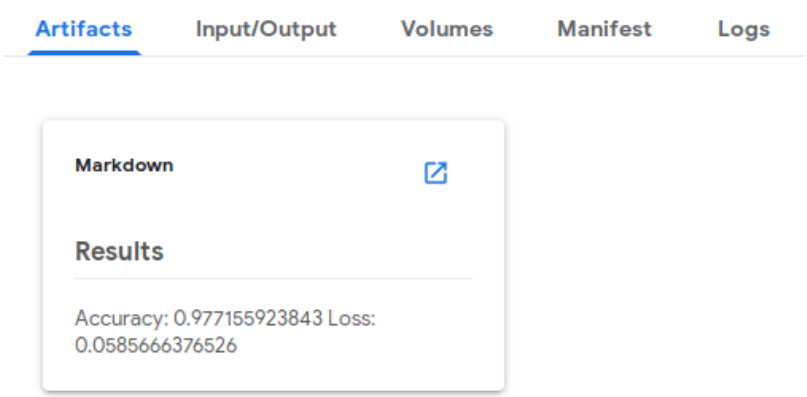


Figure 4.2: Kubeflow artifact visualization

This proves that the pipeline has been finished and can be used over and over again until a satisfying result comes out. Also continously improvement is enabled, because Kubeflow has a function integrated to sequentially repeat a run. When the data is continuosly updated this repeating run of the pipeline also improves the model with every new run, because of new data. The developer does not have to care about anything with that, because the model training is fully automated with Kubeflow.

5 Discussion

In this chapter, the results described in chapter 4 will be compared and discussed. In doing so, the criteria defined in chapter 3.1 will serve as a basis to evaluate the scope, usability, flexibility, and simplicity of the different approaches for building a pipeline. After that, an outlook will be given, in which future opportunities with such technologies and potential for improvement will be discussed.

5.1 Pipeline comparison

During this work, two frameworks - Azure Machine Learning service and Kubeflow - has been used to build an example pipeline. Additionally, these frameworks have been evaluated and tested for additional required functionalities for fulfilling the needs of DevOps for Machine Learning described in chapter 2.5. The results of these tests will be evaluated in this chapter on the basis of chapter 3.1.

For this the DevOps sets and practices should be called in rembrenence. We can see in figure **??**, that there were four sets - Steer, Develop & Test, Deploy and Operate. Even if the main focus on the created pipelines are on support for the developing and testing sets, it also accompanies the whole product lifecycle.

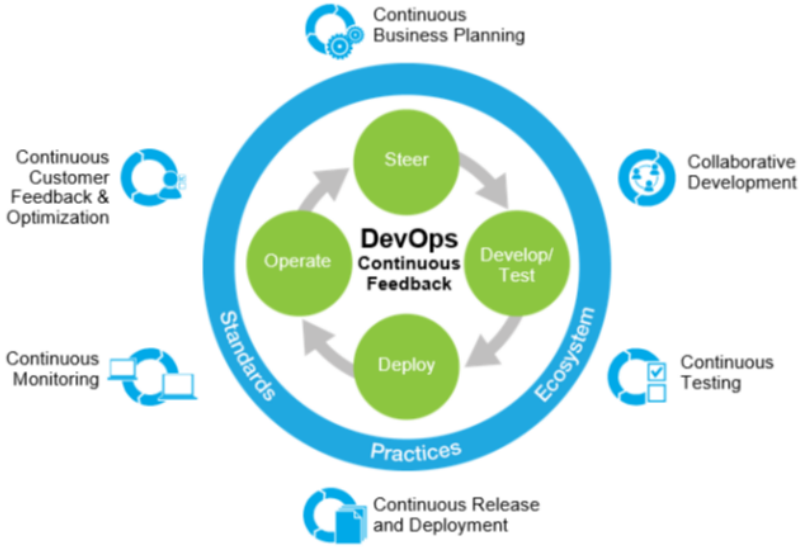


Figure 5.1: DevOps reference architecture[16]

The **steer** set was about *planning and managing* a project, which is not the task of a pipeline in first place. However, as mentioned in chapter 2.5, for Machine Learning this also includes *understanding of the data*. A good visualization is an essential point of this, which can be provided by both frameworks. Still, Azure ML service offers an easier way to this, because the loaded dataset can be directly visualized by just right clicking on the node and clicking on *Visualize*. To enable such a visualization for Kubeflow an artifact has to be created, which shows all the data in a table. This is not the purpose of an artifact in first place, but still an option, if the visualization of the data should be directly integrated in the dashboard.

The next set is about **development and testing**. There different factors are of importance. First, the factors influencing the development itself will be discussed.

The first factor is the simplicity of the frameworks so that Data Scientists can focus on their main work instead of infrastructural setups and coding. For that, the Azure Machine Learning service offers an easy way to build a working pipeline from scratch quickly. For the most basic steps of AI development, there are preexisting components defined, which can be added to the pipeline by simply dragging and dropping them to the workbench. The in- and outputs can be connected by connecting the input of one component with the output of another. Also, the configurations can be made directly on the visual interface. So there is no need for coding

experience at all, and the interface is easy to use.

Kubeflow, on the other hand, needs some knowledge of Kubernetes deployments to deploy the Kubeflow framework on Kubernetes. This can cause several problems, for example with the DNS resolution, some crashing pods or unavailable ports as the preparation of this project pointed out. After the preparations are done or an available Kubeflow deployment can be used, it offers some predefined pipelines, which can be used for similar tasks as its original purpose. However, if the developer wants to build an individual pipeline, this pipeline needs to be built from scratch. This requires coding skills because the pipelines are written in Python with the Python SDK. If the developer also needs specific components as Docker containers, which are not publicly available, these need to be coded as well. Additionally, Docker skills are required to build the containers of these components. Compared to the Azure Machine Learning service, all this is more complicated and time-consuming.

Another factor to be included in the evaluation was the option to adapt it to new frameworks and technologies enveloping over time. In the case of the Azure Machine Learning service, this completely depends on Microsoft's support for these, because the whole environment depends on the Microsoft Azure cloud. This means that the developer does not have to care about its deployment, technologies, security, and efficiency, but can not decide what to use for himself.

Kubeflow mainly depends on two technologies - Kubernetes and Containerization. Which technologies, ML frameworks or tools to use beyond that is free choosable by the developer, because the components can be created completely independent from Kubeflow, so every upcoming technology can be used without worrying about compatibility.

Also, both approaches offer the opportunity to pass on some parameters to influence the data preparation, building, and training of the model. In case of Kubeflow, these parameters can be defined by the developer, in case of Azure ML service these are predefined, so not everything is configurable.

The visualization of the results is good in both cases. Azure ML pipelines offer a visualization directly on the interface and give every necessary information as can be seen in chapter 4.1. The visualization of Kubeflow has to be enabled by adding an artifact to the corresponding

stage. The way of visualizing the results is then up to the developer.

Next, the scope of functionality has to be compared. Azure ML pipeline offers some predefined components, which makes it very easy to use. However, in exchange for its simplicity, there are not enough components to fulfill every possible need. For this Azure ML offers the possibility to insert Python scripts as a component. Still, the usage of those is not as flexible as it has to be to enable the developer to build the pipeline flexibly. Also, the in- and output of the components is very strict, and the model architecture can not be chosen freely. This can lead to some trouble, for example, because the Classifier model is not compatible with multiple columns as labels. This eliminates the opportunity to encode the categories with *One-Hot-Encoding* and aggravates the performance of the resulting model.

Kubeflow, on the other hand, offers full flexibility and functionality, because the components can be created in the way the developer chooses. This requires some coding skills of the developer but enables a flexible pipeline without any compromise of functionalities compared to local building and training a model.

Next, possibilities of *collaborative development* and *continuous integration* will be evaluated.

Both frameworks have the possibility of using an IDE to handle the necessary code. In case of Azure ML service, this can be either done on the visual interface itself or via the Visual Studio SDK for Azure ML service. Kubeflow offers the opportunity to directly integrate and collaborate with Jupyter Notebooks running on Jupyter Hub. Alternatively, the components can be coded as containers in whatever IDE the developer prefers.

Another point is the support of data collaboration platforms so that it can be worked together on data as it is already possible with code. In Azure, the datasets are usually stored directly on their servers and can be uploaded. However, it is also possible to create nodes, which connects to data services like Hive Query or Azure Blob storage. With some of these tools, versioning and collaborating on the data is possible.

Because of Kubeflow components can be written freely, it is also possible, that it accesses data from every online data collaboration platform, like Quilt or Git LFS. This also enables the possibility to versionize the data.

Regarding *Continuous testing*, in both frameworks components can be added, which directly and automatically test the resuling models every time a new model is built. The results can directly be visualized as mentioned above. All this offers quite a good opportunity to continuosly testing the models.

The third set was all about **delivery**, which includes deploying and scaling the resuling application.

An easy, automated deployment is possible in both cases by automatically creating the model, which can then be stored and accesses via an existing application.

Also, the factor of scalability is met in both cases as well. Azure ML services run on a scalable cluster on the Azure Cloud. This cluster scales automatically depending on the necessary resources for each step. Kubeflow, on the other hand, runs directly on any Kubernetes cluster. This enables scalability, as well. The resources can even be specified by the users if need be.

Last, **operation** of the resulting model is of a high importance for fulfilling the need of DevOps.

While neither of both offer opportunities to monitor the usage of the model directly on the pipeline, both provide the possibility of *continuously refitting the model*. Because of the automated steps these pipelines are running through, the developer is released of the burden to repeat running all those tasks manually when the model should be refitted. The pipelines can be run sequentially, so that as soon as the data has been updated, the model can be

All in all, these two frameworks are built for different use-cases, and both serve their purposes well. Azure ML service is for a quick building of a pipeline, which only supports the most basic steps. So it is a good way to choose if some uncomplicated model should be built, which does not need a lot of special configurations.

Kubeflow, on the other hand, offers the developer a free choice of tools and frameworks being used, so that there is no loss in functionality. Additionally, it offers the user to set a period of time, in which the pipeline will be triggered automatically. This enables a good way to continuously improve the model by continuously updating the dataset and automatically rebuild the model sequentially.
In return, the building effort is even a bit higher compared to local development. This is why

using Kubeflow pays off if some components can be reused, and not everything has to be built from scratch over and over again. However, if a long term project needs similar components for several models, which should be continuously improved, the effort to build this pipeline may be worth it.

5.2 Outlook

In the future, Machine Learning will be more and more important, and more and more complex systems will be built. This requires automated development and delivery processes, which is the reason why DevOps for Machine Learning will arise even more over time.

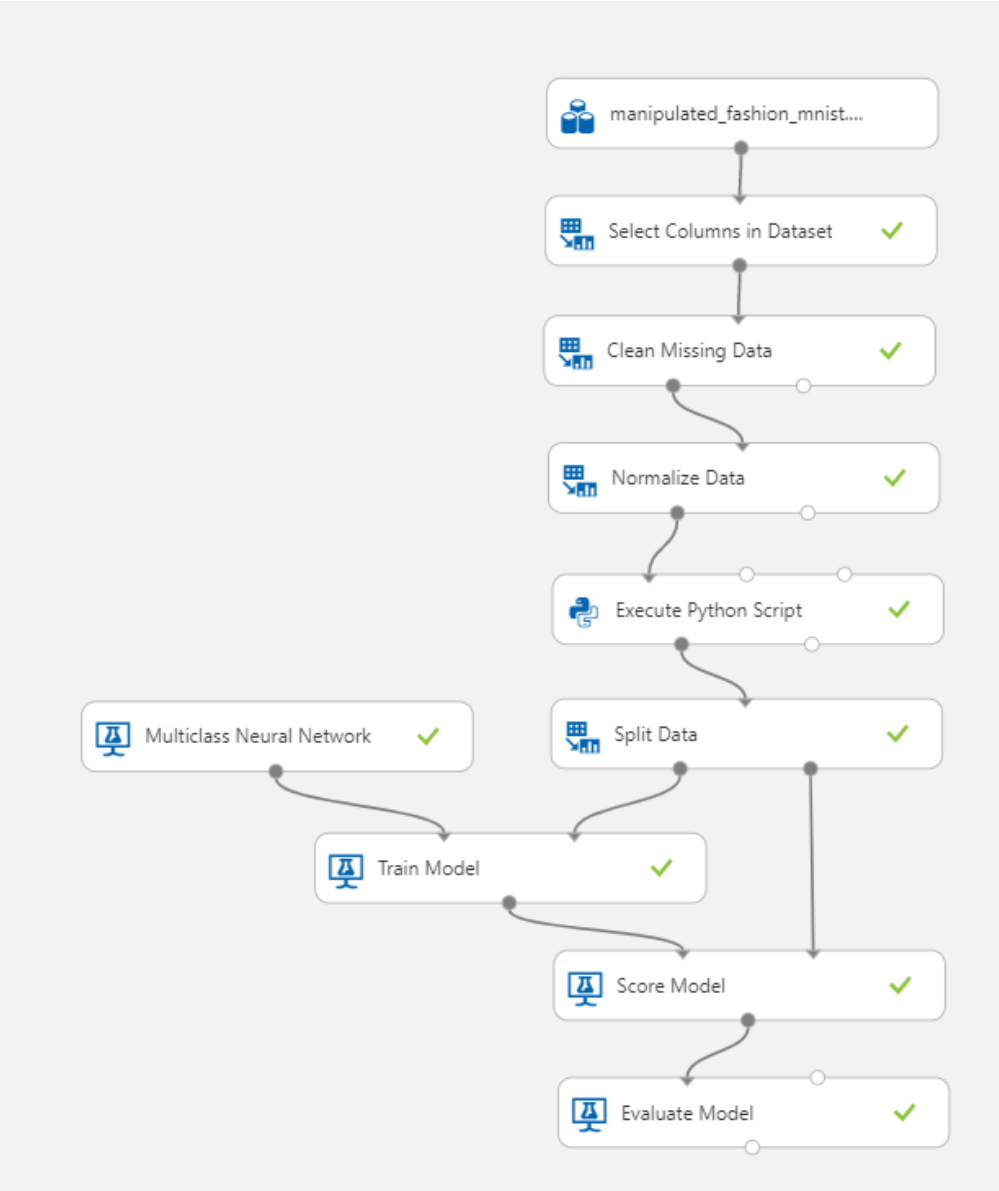
ML projects in the future need to be even more scalable because the demand and the complexity of AI will rise, which needs even more resources. At the same time, the development of such models will be getting more complex because of the rising possibilities of AI. This needs practices to keep up with collaborative, organized, and flexible development as there is already for common software development.

Kubeflow or Azure Machine Learning service can be a part of the solution in the future to automate the development and delivery processes. However, both do not offer a complete solution because things like a collaborative data platform are supported but note directly integrated. Additionally, both systems have their problems to deal with. In case of Azure Machine Learning service, this is missing functionality and flexibility, in case of Kubeflow flexibility and functionality is given but in return, the development of such a pipeline is still comparatively complex, and the system is not too easy to handle.

This is why there is no final solution for DevOps for Machine Learning yet. There are still a lot of things to do, like combining good data and model versioning with an easy way to automate the development and deployment. Kubeflow is on a good way to handle most of the things, but the system is still pretty complex. However, Kubeflow is still in development and far from finished, so there is a good chance, that it will evolve as a good possibility to support big Machine Learning projects in the future.

6 Appendix

6.1 Azure pipeline



6.2 Kubeflow pipeline implementation

```
def pipeline(
    dataset_location='/mnt/data/manipulated_fashion_mnist.csv',
    test_size=0.3,
    random_state=42,
    input_shape_height=28,
    input_shape_width=28, use_pretrained_model='False',
    model_units_num=128
    model_outputs_num=10,
    model_activation_func_layer2='relu',
    model_activation_func_layer3='softmax',
    optimizer='adam',
    loss='binary_crossentropy',
    metrics='accuracy',
    num_epochs=10,
    location_prepared_dataset='/mnt/data/prep_fashion_mnist.csv'
        ,
    location_improved_dataset='/mnt/data/impr_fasion_mnist.csv',
    location_training_images='/mnt/data/train_img.csv',
    location_training_labels='/mnt/data/train_labels.csv',
    location_test_images='/mnt/data/test_img.csv',
    location_test_labels='/mnt/data/test_labels.csv',
    location_base_model='/mnt/model/base_model.h5',
    location_trained_model='/mnt/model/trained_model.h5',
    location_result='/mnt/result.txt'):

  data_preparation = data_prep_op(dataset_location,
      location_prepared_dataset).apply(onprem.mount_pvc("fashion-
      mnist-vol", 'local-storage', "/mnt"))
  feature_engineering = feature_eng_op(data_preparation.outputs[
      'output'], location_improved_dataset).apply(onprem.
```

```
        mount_pvc("fashion-mnist-vol", 'local-storage', "/mnt"))
27    data_split = data_split_op(feature_engineering.outputs['output
        '], test_size, random_state, location_training_images,
        location_training_labels, location_test_images,
        location_test_labels).apply(onprem.mount_pvc("fashion-mnist
        -vol", 'local-storage', "/mnt"))
28
29    with dsl.Condition(use_pretrained_model == 'True'):
30      model_building = model_download_op(input_shape_height,
          input_shape_width, location_base_model).apply(onprem.
          mount_pvc("fashion-mnist-vol", 'local-storage', "/mnt"))
31      model_training = model_train_op(data_split.outputs['
          train_img'], data_split.outputs['train_label'],
          input_shape_height, input_shape_width, model_building.
          outputs['output_model_loc'], num_epochs,
          location_trained_model).apply(onprem.mount_pvc("fashion-
          mnist-vol", 'local-storage', "/mnt"))
32      model_evaluation = model_eval_op(data_split.outputs['
          test_img'], data_split.outputs['test_label'],
          input_shape_height, input_shape_width, model_training.
          outputs['output_model_loc'], location_result).apply(
          onprem.mount_pvc("fashion-mnist-vol", 'local-storage', "/
          mnt"))
33
34
35    with dsl.Condition(use_pretrained_model == 'False'):
36      model_building = model_build_op(input_shape_height,
          input_shape_width, model_units_num, model_outputs_num,
          model_activation_func_layer2,
          model_activation_func_layer3, optimizer, loss, metrics,
          location_base_model).apply(onprem.mount_pvc("fashion-
          mnist-vol", 'local-storage', "/mnt"))
```

```
37      model_training = model_train_op(data_split.outputs['
           train_img'], data_split.outputs['train_label'],
           input_shape_height, input_shape_width, model_building.
           outputs['output_model_loc'], num_epochs,
           location_trained_model).apply(onprem.mount_pvc("fashion-
           mnist-vol", 'local-storage', "/mnt"))
38      model_evaluation = model_eval_op(data_split.outputs['
           test_img'], data_split.outputs['test_label'],
           input_shape_height, input_shape_width, model_training.
           outputs['output_model_loc'], location_result).apply(
           onprem.mount_pvc("fashion-mnist-vol", 'local-storage', "/
           mnt"))
```

6.3 Kubeflow pipeline

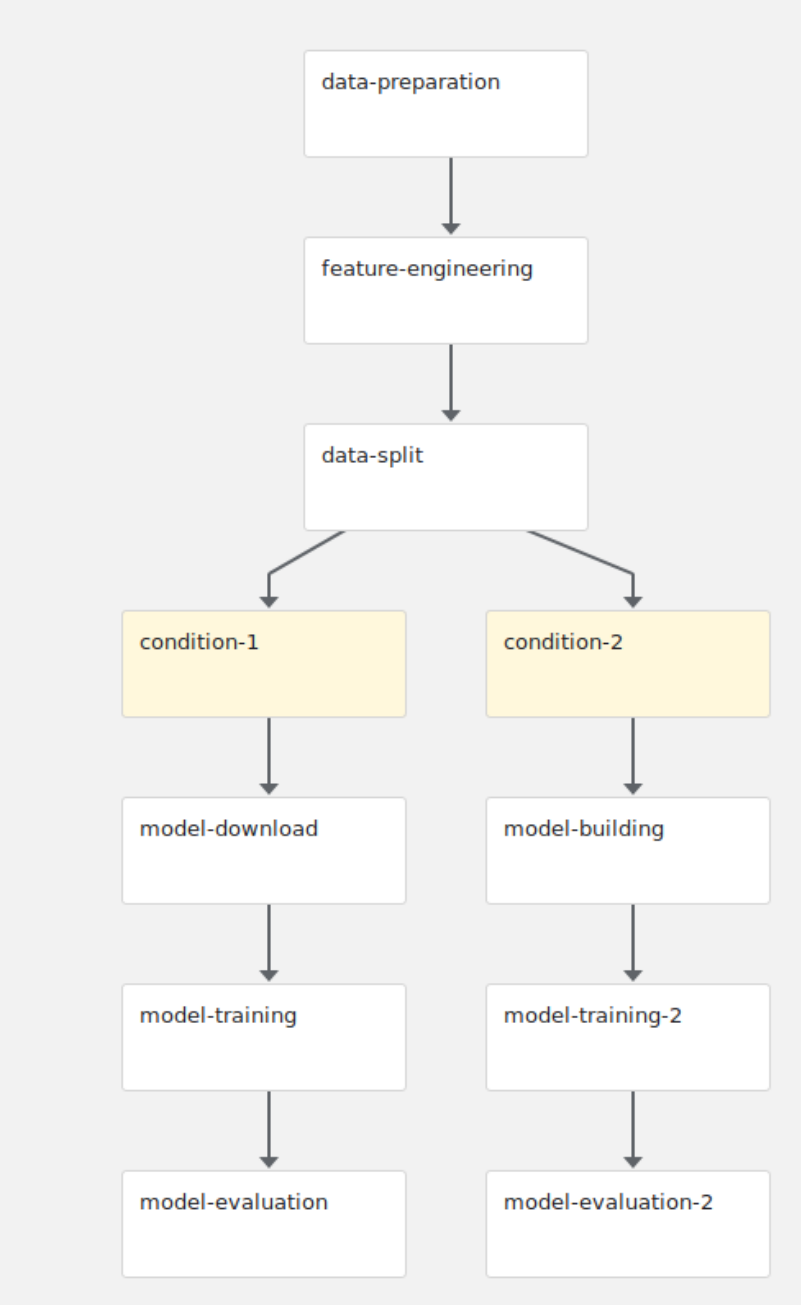


Figure 6.1: Kubeflow pipeline

6.4 Kubeflow parameter

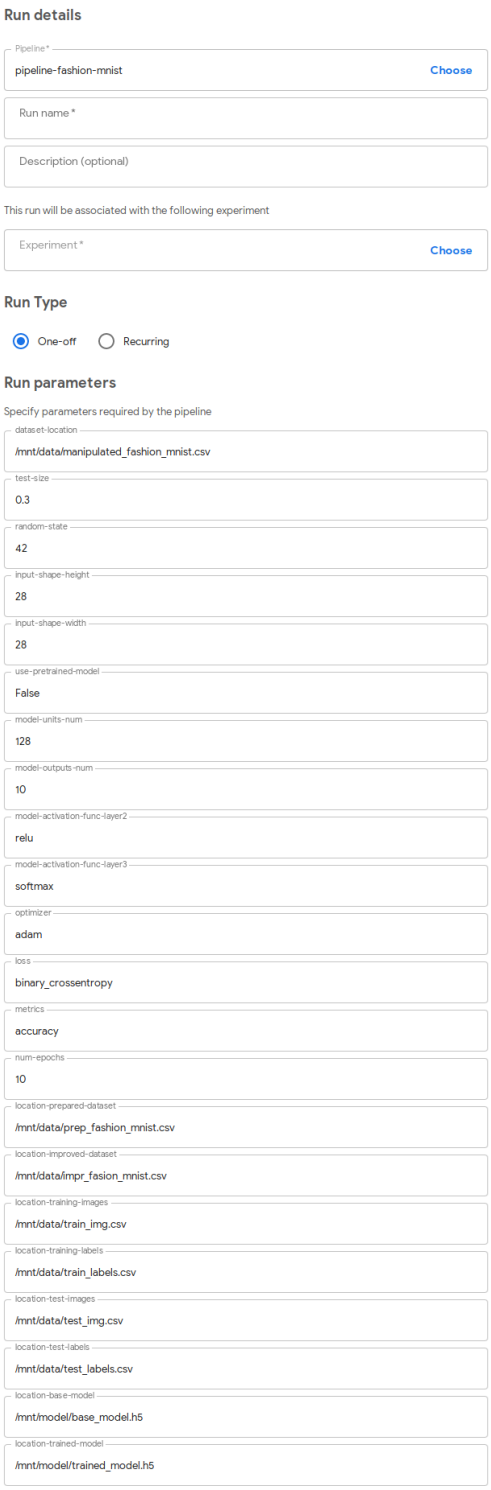


Figure 6.2: Kubeflow parameter

Literature

1. JANAKIRAM MSV. Three Factors That Accelerate The Rise Of Artificial Intelligence. *2018-05-27* [online]. Available from: www.forbes.com/sites/janakirammsv/2018/05/27/here-are-three-factors-that-accelerate-the-rise-of-artificial-intelligence/%7B/#%7D4349a30badd9

2. MICHAEL SHIRER and MARIANNE D'AQUILA. Worldwide Spending on Artificial Intelligence Systems Will Grow to Nearly $35.8 Billion in 2019. *2019-03-11* [online]. Available from: www.idc.com/getdoc.jsp?containerId=prUS44911419

3. MARIYA YAO. 6 Ways AI Transforms How We Develop Software. *2018-04-18* [online]. Available from: www.forbes.com/sites/mariyayao/2018/04/18/6-ways-ai-transforms-how-we-develop-software/%7B/#%7D1ee109f026cf

4. ANDREJ KARPATHY. Software 2.0. *2017-11-11* [online]. Available from: medium.com/@karpathy/software-2-0-a64152b37c35

5. ARMBRUST, A. FOX, AND R. GRIFFITH, M. Above the clouds: A Berkeley view of cloud computing. *University of California, Berkeley, Tech. Rep. UCB* [online]. 2009. DOI 10.1145/1721654.1721672. Available from: http://arxiv.org/abs/0521865719%209780521865715

6. JONAS, Eric, SCHLEIER-SMITH, Johann, SREEKANTI, Vikram, TSAI, Chia-Che, KHANDELWAL, Anurag, PU, Qifan, SHANKAR, Vaishaal, CARREIRA, Joao, KRAUTH, Karl, YADWADKAR, Neeraja, GONZALEZ, Joseph, ADA POPA, Raluca, STOICA, Ion and A. PATTERSON, David. *Cloud Programming Simplified: A Berkeley View on Serverless Computing*. 2019.

7. AMAZON DOCUMENTATION. *AWS Lambda – Serverless Compute - Amazon Web Services* [online]. [Accessed 25 July 2019]. Available from: aws.amazon.com/lambda/

8. BHAGYASHREE R. How Serverless computing is making AI development easier. *2018-09-12* [online]. Available from: hub.packtpub.com/how-serverless-computing-is-making-ai-development-easier/

9. DILLON. CI/CD for Machine Learning & AI. *2018-10-22* [online]. Available from: blog.paperspace.com/ci-cd-for-machine-learning-ai/

10. DAVID LINTHICUM. DevOps is dictating a new approach to cloud development. [online]. Available from: techbeacon.com/app-dev-testing/devops-dictates-new-approach-cloud-development

11. SUSAN MOORE. *How to Get Started With AIOps* [online]. Gartner, 2019. Available from: www.gartner.com/smarterwithgartner/how-to-get-started-with-aiops/

12. PUGH, Emerson W. Origins of Software Bundling. *IEEE Ann. Hist. Comput.* [online]. 2002. Vol. 24, no. 1, p. 57–58. DOI 10.1109/85.988580. Available from: http://dx.doi.org/10.1109/85.988580

13. HOLWEG, Matthias. The genealogy of lean production. *Journal of Operations Management* [online]. 2007. Vol. 25, no. 2, p. 420–437. DOI https://doi.org/10.1016/j.jom.2006.04.001. Available from: http://www.sciencedirect.com/science/article/pii/S0272696306000313

14. STEVE MEZAK. The Origins of DevOps: What's in a Name? *2018-01-25* [online]. Available from: https://devops.com/the-origins-of-devops-whats-in-a-name/

15. SAUCE LABS. *Extent of DevOps adoption by software developers in 2017 and 2018* [online]. Available from: www.statista.com/statistics/673505/worldwide-software-development-survey-devops-adoption/

16. SHARMA, Sanjeev and COYNE, Bernie. *DevOps for Dummies*. 3rd IBM Li. John Wiley & Sons, 2017. ISBN 978-1-119-41589-3.

17. IBM. Continuous Business Planning - Connecting the dots between strategy and execution. In : *Innovate2013*. 2013.

18. AMAZON DOCUMENTATION. *What is Continuous Integration? – Amazon Web Services* [online]. Available from: https://aws.amazon.com/devops/continuous-integration/

19. TALI SOROKER. *3 Reasons Why Version Control is a Must for Every DevOps Team* [online]. Available from: https://blog.overops.com/3-reasons-why-version-control-is-a-must-for-every-devops-team/

20. MILECIA MCG. Difference Between Development, Stage, And Production. *2019-04-30* [online]. Available from: https://dev.to/flippedcoding/difference-between-development-stage-and-production-d0p

21. DEBBIE. The staging environment vs test environment: What's the difference? *2018-08-01* [online]. Available from: https://www.plesk.com/blog/product-technology/staging-environment-vs-test-environment/

22. KUMAR, K. V. K Mahesh. SOFTWARE AS A SERVICE FOR EFFICIENT CLOUD COMPUTING. *International Journal of Research in Engineering and Technology* [online]. 2014. Available from: https://pdfs.semanticscholar.org/4160/859c66dc8809a1829d2d83d69b079c545300.pdf

23. MARTIN FOWLER and JAMES LEWIS. Microservices - a definition of this new architectural term. *2014-03-25* [online]. Available from: https://martinfowler.com/articles/microservices.html

24. WIGGINS, Adam. *The Twelve-Factor App* [online]. [Accessed 22 August 2019]. Available from: https://12factor.net/

25. JERRY, Preissler and TIGGES, Oliver. Docker - perfekte Verpackung von Microservices. *OBJEKTspektrum* [online]. 2015. Available from: https://www.sigs-datacom.de/uploads/tx%7B/_%7Ddmjournals/preissler%7B/_%7Dtigges%7B/_%7DOTS%7B/_%7DArchitekturen%7B/_%7D15.pdf

26. MIKESIR87. Docker is NOT a Hypervisor. *2017-05-08* [online]. Available from: https://blog.mikesir87.io/2017/05/docker-is-not-a-hypervisor/

27. DOCKER DOCUMENTATION. *Docker security* [online]. [Accessed 22 August 2019]. Available from: https://docs.docker.com/engine/security/security/

28. LIPKE, Simon. *Building a Secure Software Supply Chainusing Docker* [online]. PhD thesis. 2017. Available from: https://hdms.bsz-bw.de/frontdoor/deliver/index/docId/6321/file/20170830%7B/_%7Dthesis%7B/_%7Dfinal.pdf

29. NOAH ZOSCHKE. Modern Twelve-Factor Apps With Docker. *2015-05-18* [online]. Available from: https://medium.com/@nzoschke/modern-twelve-factor-apps-with-docker-55dd92c832b3

30. FRICKE, Thomas. Kubernetes: Architektur und Einsatz – Einführung mit Beispielen | Informatik Aktuell. *2018-01-16* [online]. Available from: https://www.informatik-aktuell.de/entwicklung/methoden/kubernetes-architektur-und-einsatz-einfuehrung-mit-beispielen.html

31. COREOS. *Overview of Pods* [online]. [Accessed 22 August 2019]. Available from: http://coreos.com/kubernetes/docs/latest/pods.html

32. JORGE ACETOZI. Kubernetes Master Components: Etcd, API Server, Controller Manager, and Scheduler. *2017-12-11* [online]. Available from: https://medium.com/jorgeacetozi/kubernetes-master-components-etcd-api-server-controller-manager-and-scheduler-3a0179fc8186

33. MICHAEL D. ELDER. Kubernetes & 12-factor apps. *2019-05-03* [online]. Available from: https://medium.com/ibm-cloud/kubernetes-12-factor-apps-555a9a308caf

34. MITCHELL, Thomas M. *Machine Learning*. 1. New York, NY, USA : McGraw-Hill, Inc., 1997. ISBN 0070428077, 9780070428072.

35. GOODFELLOW, Ian, BENGIO, Yoshua and COURVILLE, Aaron. *Deep Learning*. MIT Press, 2016.

36. AMNAH KHATUN. Let's know Supervised and Unsupervised in an easy way. *2018-07-10* [online]. Available from: https://chatbotsmagazine.com/lets-know-supervised-and-unsupervised-in-an-easy-way-9168363e06ab

37. STROETMANN, Karl. *An Introduction to Artificial Intelligence* [online]. 2018. Available from: https://github.com/karlstroetmann/Artificial-Intelligence/blob/master/%7B/%%7D0ALecture-Notes/artificial-intelligence.pdf

38. VICTOR ZHOU. Machine Learning for Beginners: An Introduction to Neural Networks. *2019-03-05* [online]. Available from: https://towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neural-networks-d49f22d238f9

39. WIRTH, Rüdiger. CRISP-DM : Towards a Standard Process Model for Data Mining. *Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining*. 2000. DOI 10.1.1.198.5133.

40. WOLF RIEPL. CRISP-DM: Ein Standard-Prozess-Modell für Data Mining. *2012-04-09* [online]. Available from: https://statistik-dresden.de/archives/1128

41. KOTSIANTIS, Sotiris, KANELLOPOULOS, Dimitris and PINTELAS, P. Data Preprocessing for Supervised Learning. *International Journal of Computer Science*. 2006. Vol. 1, p. 111–117.

42. SUNNY SRINIDHI. Label Encoder vs. One Hot Encoder in Machine Learning. *2019-07-30* [online]. Available from: https://medium.com/@contactsunny/label-encoder-vs-one-hot-encoder-in-machine-learning-3fc273365621

43. MICROSOFT DOCS. Normalize Data. *2019-05-06* [online]. Available from: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/normalize-data

44. ADITYA MISHRA. Metrics to Evaluate your Machine Learning Algorithm. *2018-02-24* [online]. Available from: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234

45. ABADI, Martin, AGARWAL, Ashish, BARHAM, Paul, BREVDO, Eugene, CHEN, Zhifeng, CITRO, Craig, CORRADO, Greg, DAVIS, Andy, DEAN, Jeffrey, DEVIN, Matthieu, GHEMAWAT, Sanjay, GOODFELLOW, Ian, HARP, Andrew, IRVING, Geoffrey, ISARD, Michael, JIA, Yangqing, JOZEFOWICZ, Rafal, KAISER, Lukasz, KUDLUR, Manjunath, LEVENBERG, Josh, MANE, Dan, MONGA, Rajat, MOORE, Sherry, MURRAY, Derek, OLAH, Chris, SCHUSTER, Mike, SHLENS, Jonathon, STEINER, Benoit, SUTSKEVER, Ilya, TALWAR, Kunal, TUCKER, Paul, VANHOUCKE, Vincent, VASUDEVAN, Vijay, VIEGAS, Fernanda, VINYALS, Oriol, WARDEN, Pete, WATTENBERG, Martin, WICKE, Martin, YU, Yuan and ZHENG, Xiaoqiang. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems* [online]. 2015. Available from: http://download.tensorflow.org/paper/whitepaper2015.pdf

46. KERAS.IO. *Home - Keras Documentation* [online]. [Accessed 2 September 2019]. Available from: https://keras.io/

47. TENSORFLOW. *TensorFlow Documentation* [online]. [Accessed 2 September 2019]. Available

from: https://www.tensorflow.org/

48. KUBEFLOW. *Github documentation: Machine Learning Pipelines for Kubeflow* [online]. [Accessed 2 September 2019]. Available from: https://github.com/kubeflow/pipelines

49. ARGOPROJ. *Github documentation: Argo Workflows - Get stuff done with Kubernetes* [online]. [Accessed 2 September 2019]. Available from: https://github.com/argoproj/argo

50. MICROSOFT DOCS. *What is Azure Machine Learning service* [online]. [Accessed 2 September 2019]. Available from: https://docs.microsoft.com/en-us/azure/machine-learning/service/overview-what-is-azure-ml

51. MICROSOFT DOCS. *Use Visual Studio Code for machine learning - Azure Machine Learning service* [online]. [Accessed 2 September 2019]. Available from: https://docs.microsoft.com/en-us/azure/machine-learning/service/how-to-vscode-tools

52. PRANEET SINGH SOLANKI. Enabling CI/CD for Machine Learning project with Azure Pipelines. *2019-08-21* [online]. Available from: https://www.azuredevopslabs.com/labs/vstsextend/aml/

53. RESEARCH, Zalando. Fashion MNIST. *kaggle.com* [online]. 2017. Available from: https://www.kaggle.com/zalando-research/fashionmnist/version/4

54. XIAO, Han, RASUL, Kashif and VOLLGRAF, Roland. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *ArXiv*. 2017. Vol. abs/1708.0.