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

1 Introduction

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 2.5.

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 2.5

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.

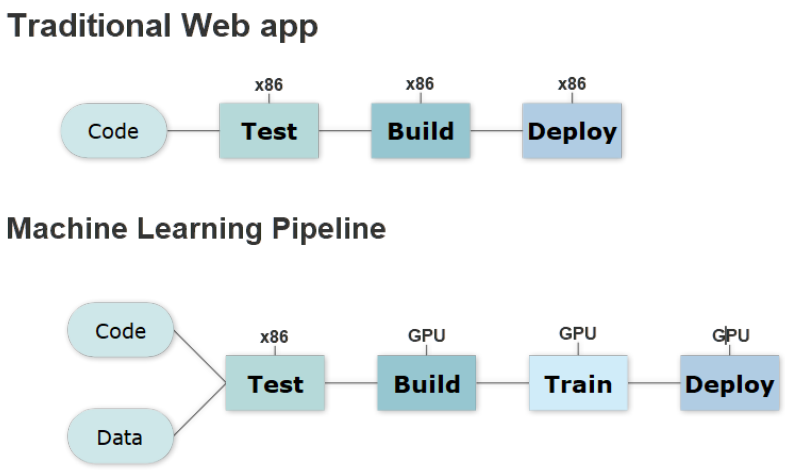

Figure 1.1: Comparison development lifecycle traditional app and ML pipeline[9]

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

In this chapter the theoretical basics, that are 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 possibilites 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 knowledges will be combined to adopt the DevOps principles explained in chapter 2.1 to the new world of AI with the help of new 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 pracitces for manufacturing was the concept of “Lean manufacturing”, which tries to reduce the waste of ressources and time of a production cycle as much as possible. With the upcoming use of Software as commercial product in the 1970s [**???**] a desire came on to create best practices for developing and operating products the same way as it was already usual in common manufacturing. [**???**]

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 is how the term “DevOps” has been created and prevailed. [**???**]

Since then DevOps has been established or at least planned in 91% of all companies as an important way to increase their efficiency of Software development. [**???**] For almost every stage of development there are principles and practices defined and continiuously improved. But before those practices will be explained, a further insight in the business need will be given.

Every process or product need a business value that cover 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 an enhanced customer experience can be guaranteed and on the other hand the efficiency of the production cycle can be increased.

One example for an 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 to two groups of randomly chosen users. Both can give their feedback to the producer and the set with the better feedback will then later be enrolled for every user.

The efficiency can be increased through reduced waste and rework because of practices to write reusable components. Another example 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 th esystem 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 important 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 alwways be captured and analyzed to provide an early warning system about potential issues and risks.

The last principle is to *amplify feedback loops* with the goal to enable a quick reaction to issues. For this organizations need to create a communication channel to its 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 splitted in four different sets based on the different periods of a product lifecycle. To each set there are several pracices, 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 **??**.

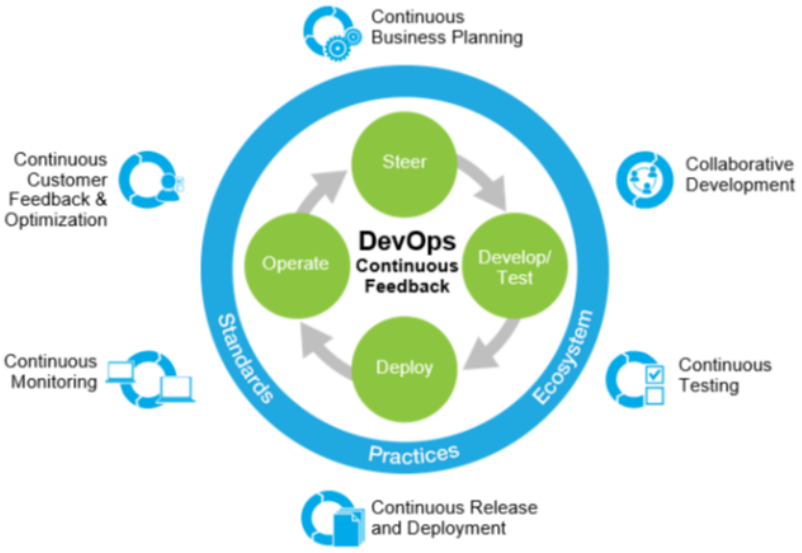

Figure 2.1: DevOps reference architecture[**???**]

The first set is called **Steer**, which is about managing and planning the development and

lifecycle of a product. This includes establishing and continuously adjusting business goals. The process resolving this issue is called *Continuous Business Planning*.

This 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. [**???**]

This is mainly done via tools and practices to track the status, feedback and needs of a project in an efficient way. First, a vision of the projects overall objective should be created and every action should be guided by it. [**???**]

The strategy, which should lead to this vision, has to be monitored and agjusted continiuously based on new information and feedbacks. This is called *continuous improvment*. For that a good dialog between a Business and IT is necessary for defining good scopes and priorities. [**???**]

Then a good plan can be built,for example with a release roadmap, which determines which feature should be completed at what time. This 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 each single feature has to be continuously tracked, so that risks will be recognized as early as possible to increase the available time to react. [**???**]

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 practictioners - architects as well as analysts, developers, specialists etc. - to work together on one project. For that it provides a common 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. Usually this is done using a version control system like Git. How a Git workflow looks like can be seen in figure 2.2.

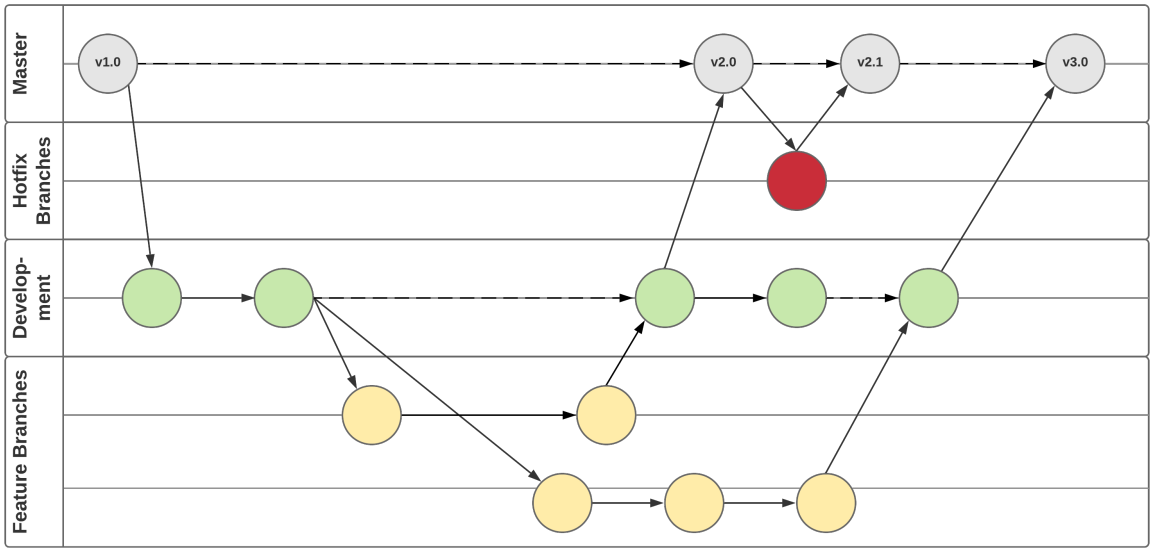

Figure 2.2: Git workflow

There the project is splitted 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. Bigger changes like added features should get an own branch. Every developer can create an own branch for a new version, so that the developers don't interfere with each other. When all code changes are done the new version will be merged to 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 to 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 descendend from the master branch.

Additionally, the application should be tested and verified continuously. For that the developer can run local unit tests to verify their changes before integrating. But this doesn't verify that

the integrated code performs as designed. [**???**] A continous integration service linke Jenkins can relieve the developer of this task and automatically builds and runs unit tests on the newly commited and integrated code. [**???**] This process is called *continuous testing*.

Another important 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 leads to quickly given feedback and enables a faster reaction to this. [**???**] This is done 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 montoried 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 includes the behavior how they are using the system as well as feedback that the users provide.

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 as well as use this knowledge for new products, that will be developed in the future.

2.1.3 Technologies

One technology to allow developers to follow above practies is **Infrastructure as code**, which enables organizations to deploy their environments faster and on a higher scale. This is done with machine readable definitions and configurations. Based on them the machine can

provide the necessary environment automatically to enable continuous delivery.

The most important technology for DevOps may be a **delivery pipeline**. 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.

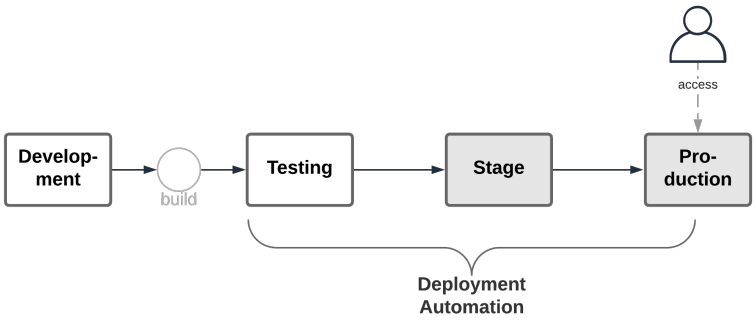

Figure 2.3: Delivery pipeline[**???**]

For every stage there is usually one specific environment. Those are represented as box. The dark boxes represent a production like 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 to also 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.[**???**]

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.

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

The delivery pipeline consists out 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 collabiration 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. But 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 providers platform and is accessible via the internet on demand. This way the end user can access the software from anywhere anytime without the need to deploy, install or maintain anything by himself.

However, the development and deployment of portable, resilient applicatons that will thrive in cloud environments is different from traditional development. Because monolithic applications need to be completly rebuilt 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 built those applications not as one monolithic software but as a suits 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 [**???**].

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 life time, projects are completed as soon as a specified set of functionalities is implemented. After that projects would be supported by a operation team. 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 are 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 leads to a more flexible environment.

The same applies to the data management, which is decentralized as well. Each service can manage an own database, which avoids problems through different conceptual models. While in centralized data management changes are usually made via transactions, in a decentralized data mangement system there is a transactionless coordination between the services. This does not necessarily results in a 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 with the problems this eventual consistency could cause are dealt with by compensating operations.

The next characteristic is an automation of the infrastructure. This includes automated testing and deployment. It is important to test every single Microservice intensively, because the operational landsacpe 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 failured quickly and automatically restore it in case an error happens. This demands a real time monitoring. To ensure this functionality, some companies are even executin tests in production by purpose to observe the behavior of the system if one service fails.

The last characteristic defined by Fowler is the evolutionary design. This 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 are mainly caused by the possi-

bility to treat every component as a single. This improves the scalability of the application as well as the productivity and speed of the development, because those independent services are faster to develop. Additionally it is easier to maintian 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 common 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 exactly 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 seperated into building, releasing and running
- Processes - Apps should be executed in one or more statless processes
- Port binding - Services should be exposes by listening on a specifeid 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

Following these guidelines stable and performant Microservices can be built. In the last years

some technologies has 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 environent, which provides everything the application needs. It is comparable to a VM (Virtual Machine), but much more light weighted. 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.

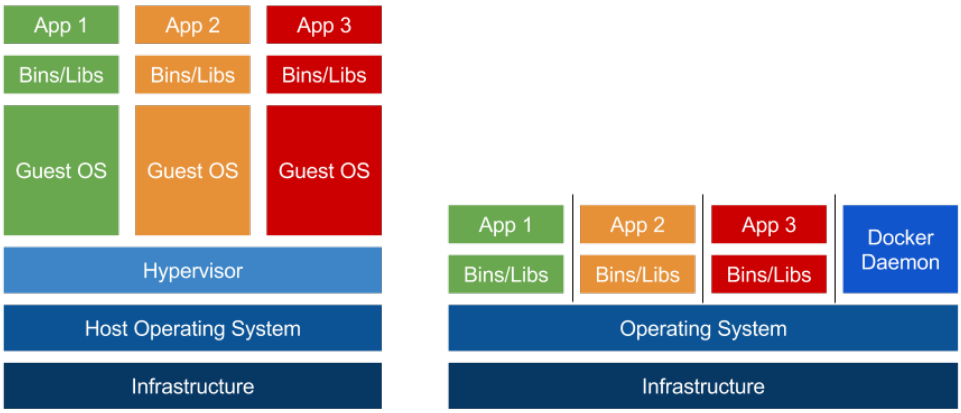

Figure 2.4: Comparison between Docker and VM[**???**]

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.

In case of a docker container those binaries and libraries are directly running in a container on the operating system without the need of a hypervisor or a complete version of a Guest OS. This 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 don't get privileged access to sockets or interfaces of other containers.

Additionally there is a Docker Daemon 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.

With this technology several of the 12 factors described above are fullfilled. One are the explicitily declared and isolated dependencies. Within the Dockerfile every dependency needs to be explicitly declared to fullfill all the requirements of the application.

Also Docker containers can't communicate with each other directly, but needs to communicate externally over with backing services over the network

Additionally Docker containers are executed as stateless processes with ephemeral storage only.

The development and production parity is given, because containers standardize how an application being delivered as well as its dependencies. [**???**]

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 every of the 12 factors are fullfilled. Instead an enabler is needed, which scalable and failure safe way to deploy those containers.

2.2.4 Kubernetes as 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.

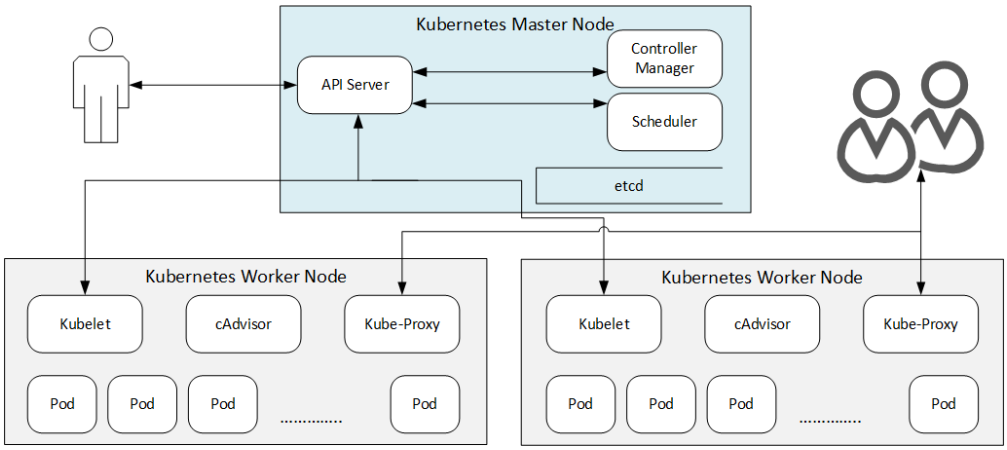

Figure 2.5: Kubernetes service allocation[**???**]

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.[cmp.[**?**]]

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

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

The Controller Manager is a daemon, which embeds all of the Kubernetes controller. 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. {cmp.[**?**], [**?**]}

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 on the basis of available resources of the pods. [cmp.[**?**], [**?**]%??

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

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. Afterwards 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.[cmp.[?], [?]]

The cAdvisor measures the usage of CPU-resources as well as demanded memory on the node, on which it is located, and notifies the master about it. Based on those measurements the scheduler allocates the pods within the cluster to ensure the best possible allocation of resources.[cmp.[?], [?]]

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. [cmp.[?], [?]]

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 a source control of all the necessary code can be done easily using git for example.

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

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.

The stage seperation is achieved through artifact management. Once the code is commited, a build occurs and the container image is built and punlished to an image registry. These releases are then deployed across multiple environments.

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

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

Also the disposability is fullfilled, because every pod can be destroyed or started in a simple and quick way. Additionally Kubernetes will automatically destroy unhealthy pods.

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

This way Docker and Kubernetes are fullfilling every of the 12-factors, which shows, that it is a very good way to provide Microservices. This is the reason why many big companies decided to use Kubernetes as 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. In the meantime 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 yourself. Instead the developer has to enter 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 continuosly 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:

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 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 identiy those trees on that picture.

2.3.1 Tasks

For fullfilling such a task it is necessary to define this task before. 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 and assigns a category out of an appropriate set . This set of categories is limited to k, so it can be described as $\{1, ..., k\}$. This can be represented as follows:

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

An example for this is character recognition. With a given input of a picture of a character 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 appartment based on features like size, location and the amount of rooms.

Structured Output tasks ask a learning algorithm to figure out the probability distribution that has produced the data set. This is then used to provide the relationship between single examples from the data set.

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

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

This is a good 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 for 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 explicitily coding algorithms and formulas, but with training a system with given input data. For training those systems there are different approaches existing, each used for different requirements.

A first approach 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 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 faciliates 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 ont the the y-axis. The task is to identify clusters in the data.

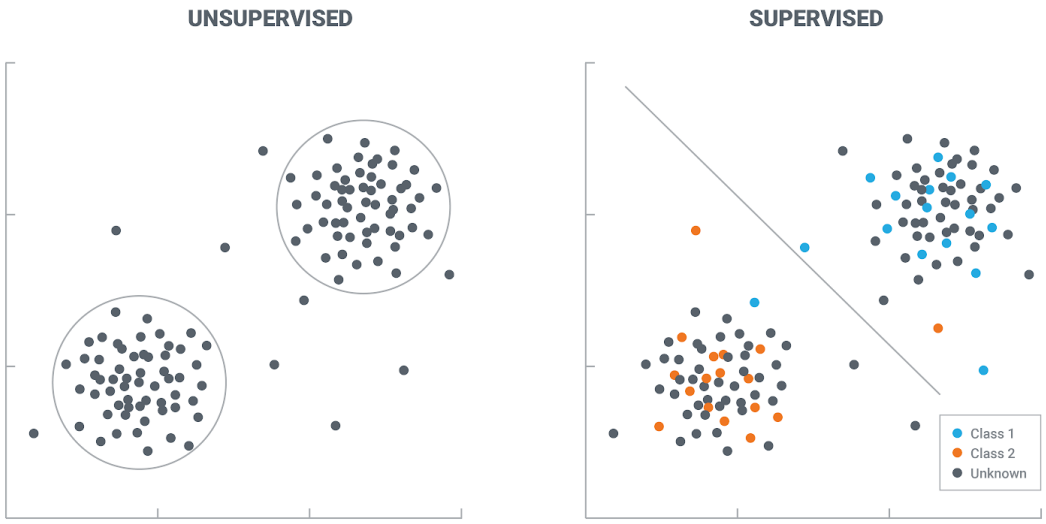

Figure 2.6: Comparison of unsupervised and semi supervised data sets used for clustering[**???**]

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 tough 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 forces 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 can't be assigned to either of the classes.

Both approaches has different situations, in which they should be the preferred way to go.
MISSING PART

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 of 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 fullfilled. 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 has 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.

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 sending their output to the output layer. This can be seen in figure 2.7.

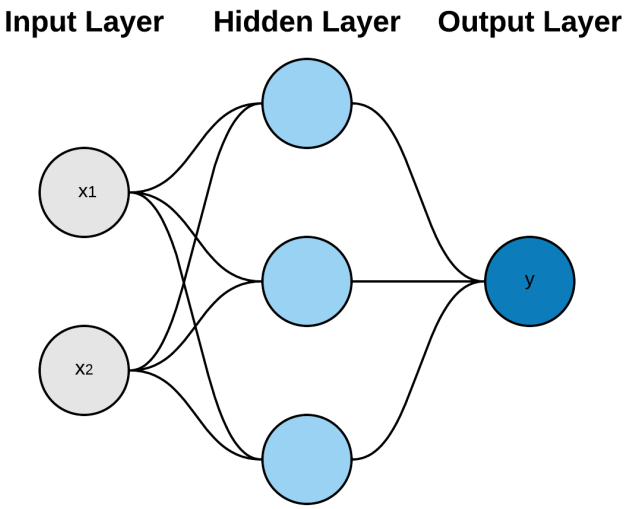

Figure 2.7: Simple neural network

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 $f(x)$

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 equation **??**

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

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

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

Finally, the activation function is being used to generate the output. For that it takes the sum

calculated in **??** as input:

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

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 in **??**

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

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 add a bias. Last it takes the result of this calculation as input for the activation function. The result of this is the output y.

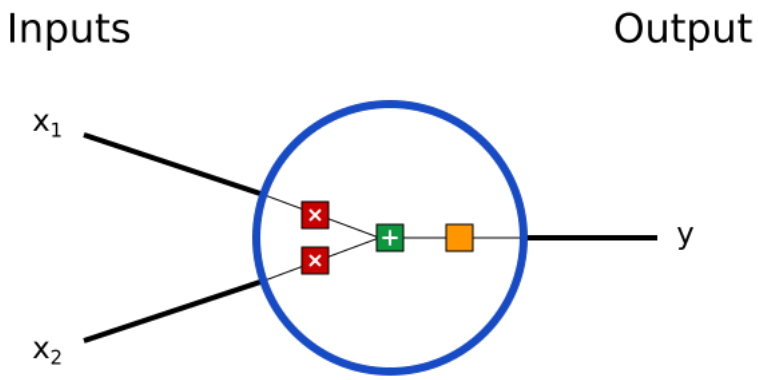

Figure 2.8: 2 input Neuron[**???**]

A neural network can consist out of more than one hidden layer. In this case they 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.

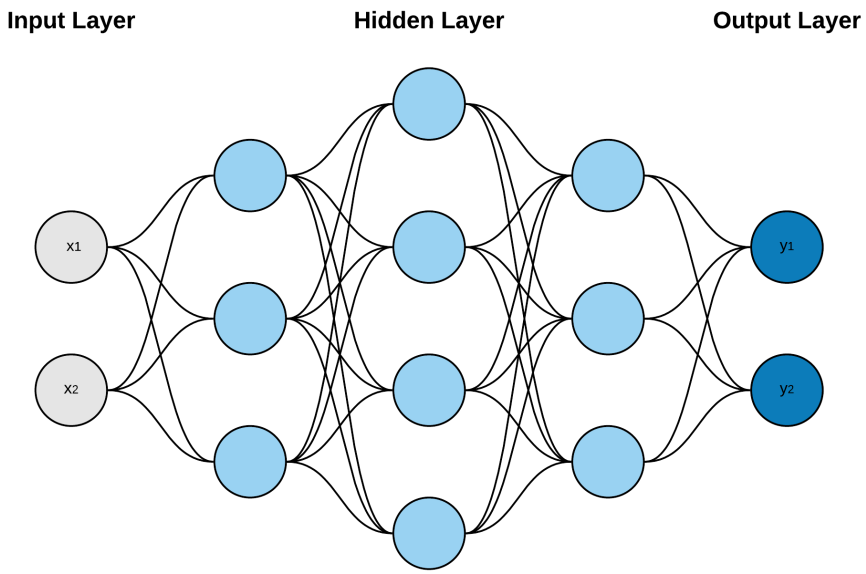

Figure 2.9: Deep neural network with 3 hidden layers

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

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

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 has 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. Overfitting means, that a network is trained with too many irrelevant features, called noise. This leads to a perfect working system for the known data, which however doesn't work at all for unknown data, because it is too specialized

on the data on which it was trained on. Underfitting on the other hand occurs if the neural network was informed by too few features or regularized too much, which prevents the neural network from learning from the dataset. How the developer handles this difficulties and what the necessary steps are in this new kind of development will be described in chapter 2.4.

2.4 Artificial Intelligence lifecycle

The new possibilities opened by Machine Learning as described in chapter 2.3 forces 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 has 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.

Figure 2.10: CRISP-DM standard[???]

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.

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 real 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.

In the example there are far too little data for a real model and much more would be needed in a real project. But 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 the estate is categorical instead of numerical, which could cause problems when training the model.

The next step is probably the most costly one - 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.

This preprocessing step includes detecting and removing noisy and redundant data. Noisy data means data, that are irrelevant for the chosen business objective. In the given example the age of the buyer is an unnecessary information, because it doesn’t change the price of the estate. That’s 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 appartments could be overrepresented, because there are twice as much appartments in the selected data than houses. The solution would be to add some more houses in the dataset or remove some appartments.
Next missing values have to be handled, because null values could influence the model in a negative way. A better way would be to fill the missing cells with average values for this type of data. But 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.
Lastly, 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.[**???**]

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. For example new features could be created based on the knowledge of the data. A condition for that is to have a really good 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 important insight of the payed price. Another meaningful feature could be some relation between the size of the real estate and the number of rooms.

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 exaple, a range from 0 to 1.

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 the modelling. 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 choosen 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.

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 a deeper knowledge. This is how a model can be continuosly improved until all stakeholders are satisfied by tuning the hyperparameters, choosing different models or improved data and features or applying different training methods.

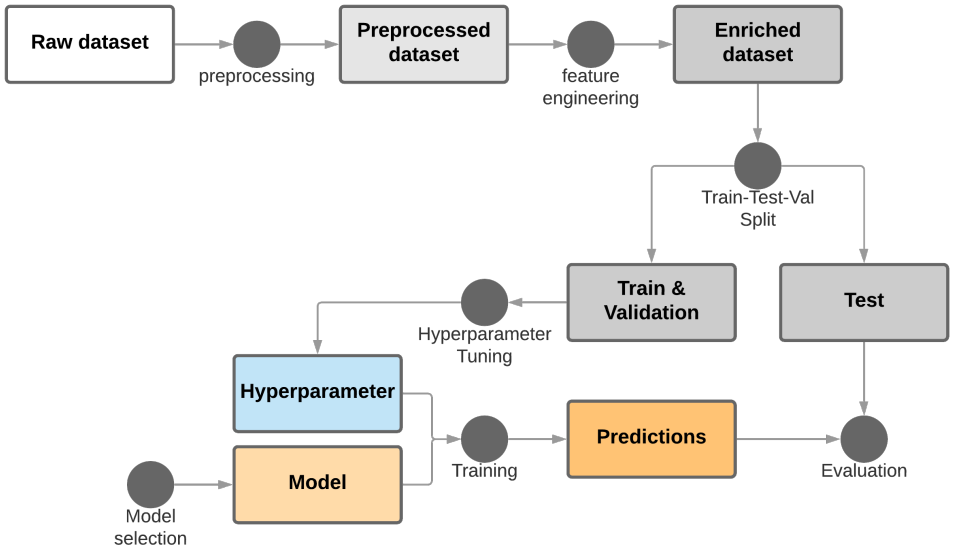

Figure 2.11: Iterative Machine Learning lifecycle

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 splitted 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 models 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 needs 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.

ASUM?

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 not only to change their development cycle, but also the operation. Additionally, new architectural models like micrsoservices, 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 **??**. To repeat, these are following:

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

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 managment 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 quick as possible.

All this also applies for the development of AI and overlaps with the CRISP-DM process described in chapter **??**. This defines as first stage the *Business Understanding*, 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 tradtitional software project.

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

The first is in the roles of the people, who handle these steps. This deep 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.

A second difference is the need of a tool to visualize these data. For that the data scientists or analysts need skills for creating visualized data as quickly as possible. A very common tool for that are *Jupyter Notebooks* as can be seen in figure **??**.

First the data need to be CODE STUFF with python, then it can be visualized easily. This leads

to a better understanding of the data, which helps finding issues, evaluate the quality, defining the way how to prepare and the objectives of what to do with the data.

However, the 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 splitted in two stages - data preperation 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, important input. This leads to several differences for 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 no designed for handling other data than code. This results in the need of another tool to handle big datasamples 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 for this problem could be Git **LFS!** (**LFS!**), which is desinged for storing audio samples, graphics, datasets and videos.

IDEs - jupyter notebook?

Continuous testing

delivery

operation - monotir, feedback, refitting => more flexible deployment

Another eminent difference are the roles of the developing people. While in traditional development the developers are IT specialists, who can handle the operation 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 special skillset when it comes to the preparation of data and feature engineering, but sometimes they are lacking in experience with operation tools like delivery pipelines. That's why an easier 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.

evaluate principles - production like (k8s); repeatable and reliable (pipeline); monitor and validate (harder; get user feedback; can be reused)

pipeline

3 Method

3.1 Catalogue of criteria

3.2 Used frameworks and libraries

3.2.1 Kubeflow

3.2.2 Tensorflow

3.2.3 R-CNN

3.3 Creating the necessary environment

3.3.1 Minikube

3.3.2 Kubeflow

4 Result

4.1 Kubeflow pipeline

4.2 Azure pipeline

4.3 Other framework?

5 Discussion

5.1 Pipeline comparison

5.2 Outlook

6 Appendix

placeholder

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. *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/