Leveraging Kubernetes for Machine Learning with Kubeflow

Marcel Juschak - marcelj@kth.se

I. Introduction

Nowadays, there is an accelerating importance and build up of IT infrastructure, referred to as ”the cloud”. The technology landscape consists of systems that are ubiquitous, distributed and highly dynamical. This is particularly the case for machine learning (ML) applications that are trained on real-time data streams from sources like satellites, social networks and online surfing, while at the same time serving predictions from machines that are replicated across the globe. The process of training and deploying an ML model introduces additional complexity, as can easily be seen from the typical ML pipeline shown in figure 1. Incoming data streams have to be validated, repaired and then used for training a model on a sufficiently large computing cluster. The model itself has to be validated and compared against other models, e.g. with A/B testing. The final model has to be deployed and monitored until it is finally replaced with a new one.

All of this complexity raises the question of how an ML system can be designed such that it is (1) agnostic w.r.t. the specific environment (e.g. changing IP addresses and hardware), (2) composable so that specifically required libraries can be used at any step of the process and (3) automated to increase reliability and efficiency.

In the last decade, DevOps Tools like Docker emerged that provide standardized *containers* in which software can be packaged before deploying them. These containers provide the consistent environment the software needs and decouples them from the actual hardware and environment. Orchestrators like Kubernetes help in replicating services (containers) in different regions to achieve availability and robustness. At the same time they provide mechanisms like DNS lookups by service name instead of IP address to deal with rapidly changing environments. Finally, systems like Kubeflow build upon Docker and Kubernetes but capture the nuances of the ML field by adding additional tools. Even more important, Kubeflow makes all of the advantages of Kubernetes usable for ML practitioners that are not experts at it. In the following sections, we will have a closer look at the fundamental DevOps tools Docker and Kubernetes. After that, we will see how Kubeflow makes these technologies easily usable for the ML workflow.

II. Docker

Docker has become one of the most important and loved tools for developers. Its main goal can be summarized as ”moving software from machine A to B reliably and automatically” [2]. Among professionals, the popularity of Docker as

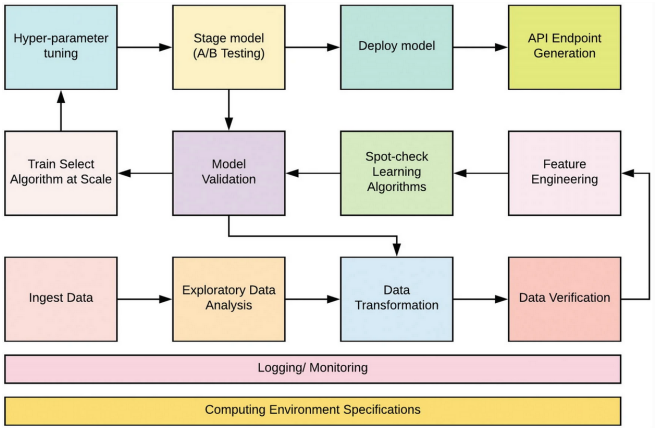

Fig. 1. Typical ML pipeline [1]

compared to other platforms in 2020 seems to be on par with Windows and Linux [3], as seen in figure 2. When it comes to which new tools developers would like to use, figure 3 shows that Docker, and notably also Kubernetes, are comparable to Git [4].

The popularity of Docker is tied to the fundamental problem it solves which is alluded to by its name [2]. The shipping industry in the middle of the 20th century was revolutionized when containers were introduced. Before, it was difficult to ensure that the dock workers at the other end of the world would handle your cargo with care and e.g. ensure that the car would not crush the piano besides. But as containers were introduced, any commodity could be packaged in the same standardized manner. Consequently, all the infrastructure around it could be standardized as well (think of ships, cranes, etc...). This separation of concerns regarding what is packaged and how the cargo is handled lead to automation and reliability. This exact revolution is repeated with the docker platform and has a comparable impact on the IT industry.

Any software assumes a certain environment like specific library versions. Docker packages all of the dependencies, including the OS, in the container so that it runs exactly the same on any hardware where Docker can be run. One may object that virtual machines (VMs) were introduced to solve this issue, so why does the introduction of docker containers have such a great impact? VMs are important, but their essential shortcoming for most use cases is that they package too much, including whole hard drives and network interfaces [2]. In contrast, containerization can be seen as lightweight virtualization [5]. The container may use a Linux distribution that is as small as 5 MB in size [6] and use storage

outside of the container in a way as if the directories were inside the container (*bind mounts*). The decisive difference is therefore mostly resource efficiency (for more details see [5]). This fits perfectly into the modern paradigm of focusing on the development of many lightweight microservices instead of few monoliths.

III. Kubernetes

In the last section, we looked at the concept of containerization which lays the foundation for many applications in the cloud. It allows for creating microservices that can be deployed anywhere in an easy and reliable manner. However, these microservices must be coordinated in order to become a fully functional system. An *orchestrator* is a tool that determines on which cluster-nodes containers should be deployed and how they communicate. Docker provides his own orchestrator *Swarm* which is remarkably easy to use and covers roughly 20% of the features that are needed by 80% of the users [7]. However, alongside Swarm, Kubernetes emerged with the aim to cover 100% of the features. Naturally, Kubernetes is not easy, but very complex and powerful. It is an open source project released by Google in 2015 with the ambition to serve as a backbone for most cloud applications [7]. It is indeed a *backbone*, as professionals usually do not use vanilla Kubernetes, but *third party extensions* that provide the specific features they need. The tool uses the operator pattern [8] to provide a means to extend the default Kubernetes API and CLI in a way as if the new commands were part of core Kubernetes itself.

Kubernetes is designed for declarative as opposed to imperative configurations [9]. In imperative configurations, a system state is achieved by entering a series of explicit instructions like create..., run..., connect... which makes it easy for the user to understand what is happening. However, the user must know the current state in order to choose the right instructions. On the other hand, in declarative configurations the user describes a desired system state and it is left up to the system to find a way to transition to it. This approach is fully in the spirit of DevOps, as it allows for more automation due to the system itself dealing with all the potential dynamics.

Because of this, most of the user's work consists of specifying .yaml files that are used by Kubernetes to continuously check whether the actual state matches the desired state. If it does not, it takes action to correct the state. If, for example, only three replicas are running, but 5 are specified, Kubernetes will automatically schedule two additional replicas to correct the state.

IV. Kubeflow

As seen in figure 1, the development process of an ML system contains many steps and it is recommended to design them as micro services in order to achieve good scaling, maintenance, re-usability of models, etc... [1]. In other words, we want to containerize our services and use an orchestrator to run them on a computing cluster. Kubeflow is a platform that helps deploying machine learning workflows on Kubernetes. Just as Kubernetes, it started out as an internal project at Google before it was made open source in 2017. Its goal is to "abstract away the technicalities of managing a Kubernetes cluster so that a machine learning practitioner can quickly leverage the power of Kubernetes" [1]. ML engineers are thus given a toolkit for the deployment of ML workflows that is both easy to use and harnesses the advantages of Kubernetes, namely portability, scalability, repeatability and composability [10]. Let us have a look at how each of these properties support the development of a ML system.

A. How Kubeflow & Kubernetes improve ML systems

Portability is a valuable property for the typical workflow of an ML practitioner. Often, experiments are performed on a subset of the data on a local machine like a laptop, as there is no network delay and the user has more control over the environment. Once the code is fully functional and final, it has to be moved to the cloud for execution on the full data set on a more powerful computing cluster which consists of several nodes. Being able to use the same code regardless of where to code runs eliminates a lot of overhead.

There are stages of the ML workflow that are either more or less resource demanding, like data preprocessing, model training, monitoring and model serving. Being able to *scale* the underlying hardware and number of nodes easily helps in allocating only the necessary resources to the current step.

There is a very rich ecosystem of ML libraries that solve different problems. *Composability* allows for integrating the "best-of-breed" libraries that fit the user's task. Because of that, Kubeflow does not aim to reinvent every library, but helps combining existing ones.

Repeatability is important in any development process. Docker containers facilitate this by being both immutable and ephemeral. This means that containers are never updated in the sense of a linux apt-get command. Instead they are rebuilt and redeployed every time something significant changes. This has the advantage that it is easy to track breaking changes to its cause and roll back to a previous version of the container [9]. The automation and documentation that comes by specifying .yaml files support the repeatability as well.

B. Kubeflow's Architecture

In figure 4 we can see the architecture of Kubeflow and how it interacts with other tools. The bottom shows how Kubernetes abstracts away all the different platforms the application may run on. Thus, Kubeflow runs anywhere Kubernetes does. The blue middle layer shows all the tools and extensions Kubeflow adds in order to integrate the standalone ML tools from the top layer into Kubernetes.

Among the Kubeflow tools in the blue box, we do not only find operators that extend Kubernetes, but also the main tools that Kubeflow offers to enhance the ML process in general. The perhaps most important of them is the pipeline, as it facilitates the main goal: end-to-end ML. The robustness and maintainability that come with the automation through pipelines is important for production. Pipelines are defined in python files but are compiled into static .yaml files that are then applied to the Kubernetes cluster [11]. Pipelines consist

of an arbitrary number of pipeline components that have two parts of code. The first part is the client code and is necessary for communicating with other endpoints to submit jobs, e.g. connecting to the Google Cloud Machine Learning Engine. The second part is the code that is actually run on the cluster. The output of one component serves as input to the next. Ekaba Bisong (2019) gives an example of such a pipeline [12]:

1) Move raw data hosted on GitHub to a storage bucket.
2) Transform the dataset using Google Dataflow.
3) Carry out hyper-parameter training on Cloud Machine Learning Engine.
4) Train the model with the optimized hyper-parameters.
5) Deploy the model for serving on Cloud MLE.

Another tool that Kubeflow provides is a user interface that is very helpful in managing all of the complex processes on large clusters. Figure 5 illustrates this UI with the above mentioned pipeline. We see that the pipeline can be visualized as an easy to understand flow-chart and that potential errors (or lack thereof) can be highlighted at the appropriate places.

V. Conclusion

In this essay we have looked at why Docker and Kubernetes are currently very popular, powerful and demanded technologies. In addition, we saw how Kubeflow abstracts away much of Kubernetes' complexity to make it easily usable by ML practitioners. Finally, we investigated the impact of Kubernetes on ML applications and looked at the additional tools Kubeflow offers to enhance the ML workflow. The newly gained insights can be summarized as the following **key takeaways**:

1) Containerization (e.g. with Docker) decouples software from its environment.
2) Orchestrators (like Kubernetes) automatically schedule container replicas across nodes, requiring only the desired system state, not instructions on how to achieve it.
3) Kubeflow integrates most common ML tools, adds its own on top to manage them and makes the power of Kubernetes available to ML practitioners.

Before writing this essay, I knew that Kubernetes is very complex, powerful and highly demanded in my field. However, with the rise of Kubeflow it became clearer to me that, becoming a Kubernetes-expert might not be a requirement. Depending on my professional environment, applying Kubernetes through Kubeflow might be sufficient.

References

[1] Ekaba Bisong. "Kubeflow and Kubeflow Pipelines". In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Berkeley, CA: Apress, 2019, pp. 671–685. ISBN: 978-1-4842-4470-8. DOI: 10.1007/978-1-4842-4470-8_46. URL: https://doi.org/10.1007/978-1-4842-4470-8_46.

[2] Solomon Hykes. *Why We Created Docker*. 2013. URL: https://www.dotconferences.com/2013/06/solomon-hykes-why-we-built-docker (visited on 04/09/2022).

[3] Stack Overflow. *Stack Overflow Developer Survey 2020*. 2020. URL: https://insights.stackoverflow.com/survey/2020#technology-platforms-professional-developers5 (visited on 04/09/2022).

[4] Stack Overflow. *Stack Overflow Developer Survey 2021*. 2021. URL: https://insights.stackoverflow.com/survey/2021#most-loved-dreaded-and-wanted-tools-tech-want (visited on 04/09/2022).

[5] Anuj Kumar Yadav, M. L. Garg, and Ritika. "Docker Containers Versus Virtual Machine-Based Virtualization". In: *Emerging Technologies in Data Mining and Information Security*. Ed. by Ajith Abraham et al. Singapore: Springer Singapore, 2019, pp. 141–150. ISBN: 978-981-13-1501-5.

[6] Docker Hub. *Linux Alpine*. URL: https://hub.docker.com/_/alpine (visited on 04/09/2022).

[7] Bret Fisher. *Docker Mastery: with Kubernetes + Swarm from a Docker Captain*. URL: https://www.udemy.com/course/docker-mastery/ (visited on 04/09/2022).

[8] Kubernetes. *Operator pattern*. URL: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/ (visited on 04/09/2022).

[9] Kelsey Hightower, Brendan Burns, and Joe Beda. *Kubernetes: Up and Running Dive into the Future of Infrastructure*. 2nd. O'Reilly Media, Inc., 2019.

[10] Kubeflow. *An introduction to Kubeflow*. URL: https://www.kubeflow.org/docs/started/introduction/ (visited on 04/10/2022).

[11] Yue Zhou, Yue Yu, and Bo Ding. "Towards MLOps: A Case Study of ML Pipeline Platform". In: *2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE)*. 2020, pp. 494–500. DOI: 10.1109/ICAICE51518.2020.00102.

[12] Ekaba Bisong. "Deploying an End-to-End Machine Learning Solution on Kubeflow Pipelines". In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Berkeley, CA: Apress, 2019, pp. 687–695. ISBN: 978-1-4842-4470-8. DOI: 10.1007/978-1-4842-4470-8_47. URL: https://doi.org/10.1007/978-1-4842-4470-8_47.

[13] Kubeflow. *An overview of Kubeflow's architecture*. URL: https://www.kubeflow.org/docs/started/architecture/ (visited on 04/10/2022).

VI. Appendix

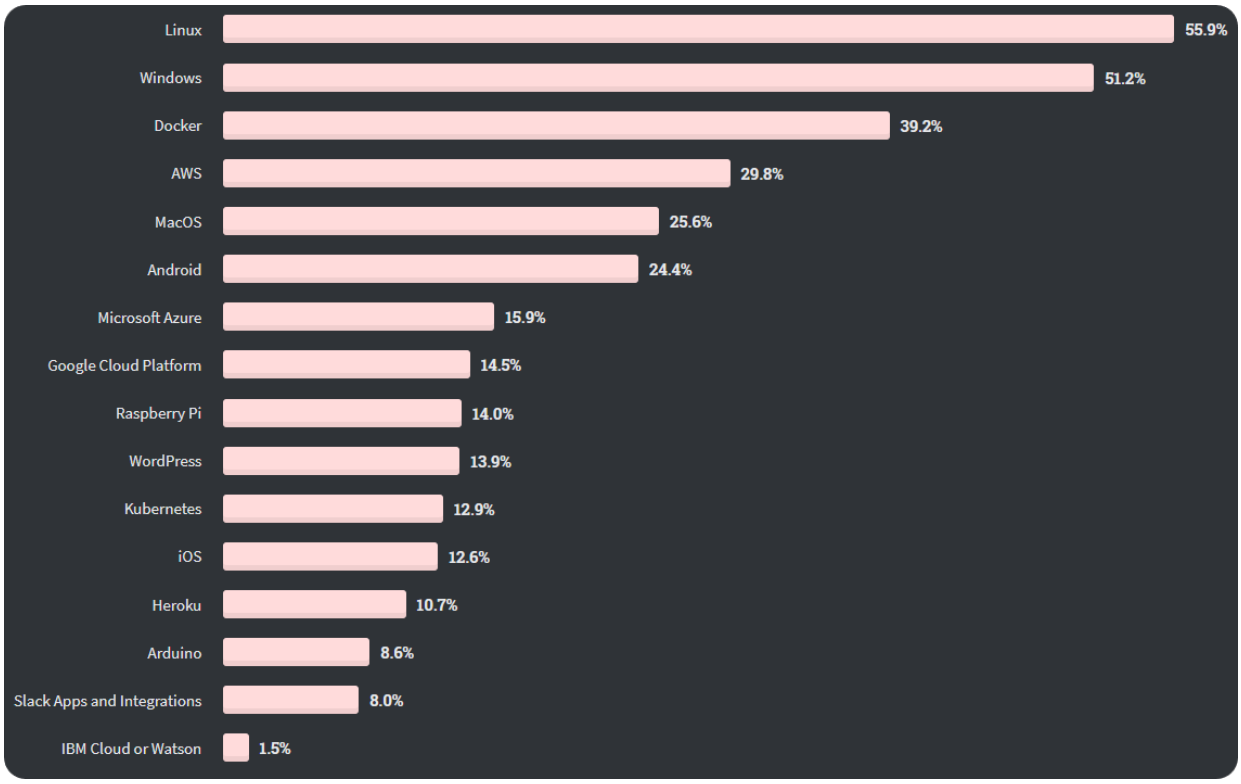

Fig. 2. Popularity of different platforms among professionals [3].

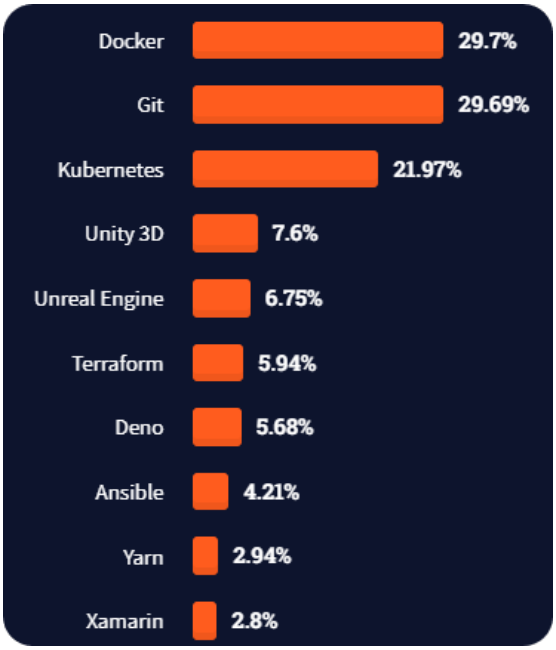

Fig. 3. Percentage of developers that want to use a specific technology they do not use yet [4].

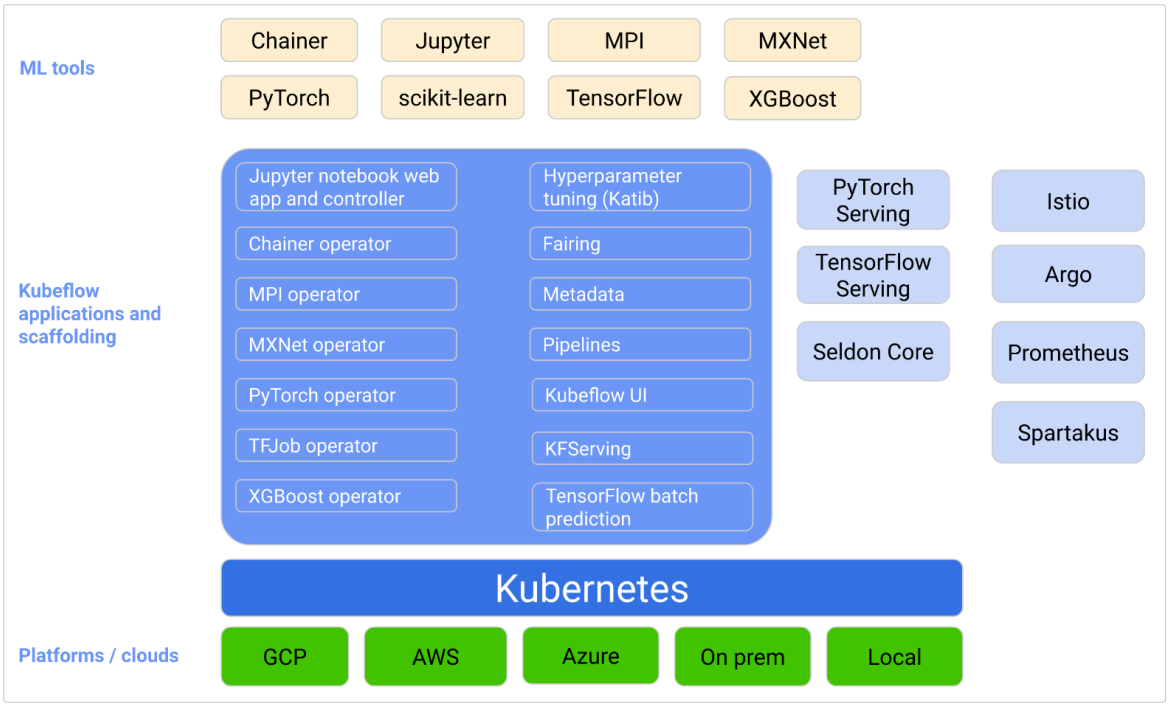

Fig. 4. Kubeflow architecture [13].

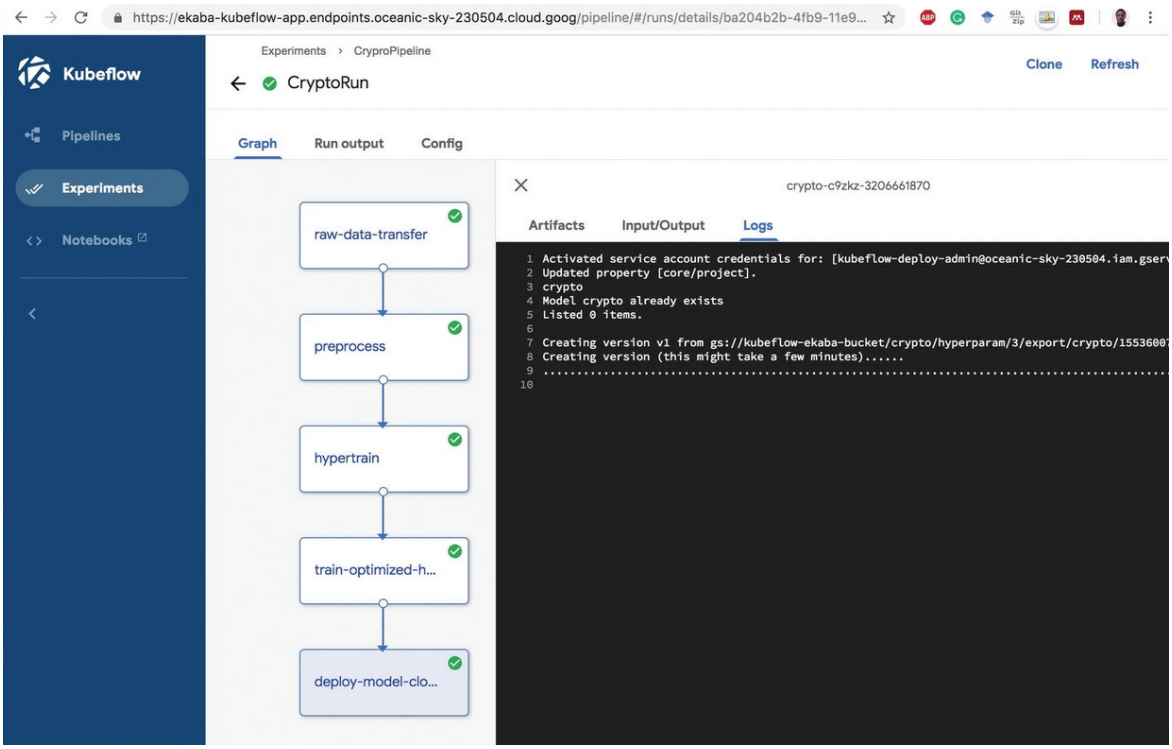

Fig. 5. Kubeflow UI showing a 5-step pipeline [12].