DD2482 Automated Software Testing and DevOps: Kubeflow or MLflow - which one to choose?

Aksel Uhr, auhr@kth.se
Nikolaos Smyrnioudis, nsmy@kth.se

May 2022

Contents

1 Introduction

MLOps is a relatively new field within machine learning and covers the same fundamental idea as DevOps. That is, automation of the entire development process from business needs to deployment of a technical solution [3]. Thus, the practice of MLOps could be considered a descendant of DevOps, whereas the main difference lies in continuously deploying machine learning models rather than software [9].

According to Columbus (2021) 88% data science projects fail. Furthermore, 18% of requested companies implied that it takes over three months to deploy a model to production [2]. Thus, the motivation behind MLOps regards addressing frequent issues which cause project failures and common problems that occur during the typical workflow of a machine learning project. For instance problems revolving around automated collaboration, communication and trust among teams as well as building, testing, monitoring and continuously deploying models [10].

As of today, there are various MLOps tools to incorporate into projects as an attempt to address the mentioned issues. Some notable ones are: Kubeflow, MLflow, Metaflow and Seldon Core. Due to many various options, the selection of MLOps tools might not be trivial for a team with regard to their needs. Therefore, this paper aims to describe and compare two widely used MLOps tools in order to facilitate the selection for teams that wish to incorporate MLOps to their project. The paper's focus will be on open source tools and the comparison will be conducted between MLflow and Kubeflow. There are also other studies comparing MLOps tools [8].

2 MLflow

MLflow is a tool that helps teams improve their MLOps workflows. Although its functionality is mainly centered around the Experimentation stage of machine learning, it also provides functionality for the deployment of machine learning models [7].

MLflow addresses some problems that in the past many machine learning teams have faced. It specifically targets the problem of not knowing which code, data and parameters were used to produce a single result. For example a team may have created a cat vs dog classifier that reaches 99% accuracy once but they may not be able to recover the exact configuration that resulted in this accuracy due to the large amount of runs with different parameters that are usually executed. Aside from that it also addresses the problem that deploying models is hard.

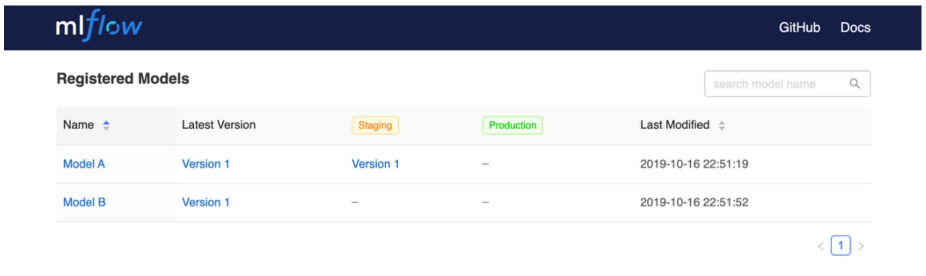
There are four components that MLflow consists of: Parameter logging, MLflow projects, MLflow models and MLflow registry.

Parameter logging offers functionality for tracking, declaring parameters such as the learning rate, number of epochs etc. for a single run and organizes several runs within experiments [7]. Metrics can also be tracked but they are distinguished from the parameters as they are the output results of each run. MLflow provides a GUI tool where users can see timestamps , parameters and metrics for their runs and experiments, as well as visualize the metrics. An interesting feature is also auto-log, which are integrationS with several popular libraries such as PyTorch-lightning. Libraries like PyTorch-lightning have logging logic of their own and MLflow allows users to use the logging logic and move their logged parameters and metrics to MLflow's GUI by only writing one line of code. Finally, the tracking server that receives the parameters and metrics can be set up at a remote endpoint, which allows multiple developers to collaborate by sharing results with a common interface [7].

The other component of the experimentation phase is MLflow projects. Using MLflow projects users can define a virtual environment such as conda, used for the project as well as entry points, scripts that can run [7]. These scripts can be scripts that download data as well as training scripts. It also supports containerized environments for even more reproducible.

Moving on the the deployment phase, the first relevant component is MLflow models. This component creates a directory whenever save_model is called which contains the saved model in multiple different formats called flavors. The model always has a python_function flavor which can take a Numpy array, a data frame or a dictionary mapping strings to Numpy arrays [7]. MLflow also provides a native integration for deploying python_functions to Amazon SageMaker and Microsoft Azure ML.

The final component, MLflow registry handles trained models. A trained model can be annotated with markdown descriptions indicating algorithms or specific details [7]. It can also be versioned and whenever a new model is pushed with the same name, the version is increased by 1. Of course, the models are hosted, can be downloaded and support CRUD operations.



Registered Models

Name	Latest Version	Staging	Production	Last Modified
Model A	Version 1	Version 1	–	2019-10-16 22:51:19
Model B	Version 1	–	–	2019-10-16 22:51:52

Figure 1: MLflow GUI of the model registry.

3 Kubeflow

Kubeflow is a platform and descendant of the popular orchestrator Kubernetes. Kubernetes provides a way of deploying container-based applications which yields higher efficiency compared to hosting on virtual machines [6]. Similarly, the idea behind Kubeflow is to build and deploy containerized machine learning models. More specifically, it concerns placing distributed machine learning components into different containers throughout the entire workflow [1]. In this context, a component is a part of the deployed model, for instance model training, model monitoring and logging once the model is in production. Training on Kubernetes brings many benefits. For example, models benefit by auto-scaling, as containers can be assigned more CPU and GPU resources according to their needs. Additionally, scaling can also happen in the instance level as Kubernetes provides automatic load balancing for pods that receive a lot of traffic.

According to Kubeflow's documentation, Kubeflow has six main components and offers external plugin integration as well [5]. The core components are Kubeflow Notebooks, Kubeflow Pipelines, Katib, Training operators and multi-tenancy. An example of an external plugin is KServe.

Kubeflow Notebooks provides a way to run web based development environments such as Visual Studio Code inside a Kubernetes cluster by running them inside pods [5]. Therefore, it is possible to create notebooks directly in the cluster rather than locally on workstations. Doing so, the model can more easily integrate with other Kubeflow components and Data scientists or machine learning engineers can access the notebook remotely. Furthermore, it is possible to leverage the computing power for data intensive machine learning models [5]. Creating a new server and notebook on Kubeflow will give you various customizable options such as ram and CPU usage, workspace limits and configurations.

Kubeflow Pipeline is Kubeflow's main component [5]. This is the core component that provides the entire end-to-end orchestration. That is, enabling and simplifying the entire automation of a machine learning pipeline. A pipeline consists of the machine learning workflow with all included components and how they combine in the form of a graph [5]. A component in a pipeline is a set of code, for instance the data preprocessing or model training. By defining the pipeline and thus the flow, the user can easily follow the different steps. Once defined, the user may execute the pipeline, a so-called run, and follow the output of it, for instance ROC curve or a confusion matrix [5]. Another feature provided by Kubeflow Pipeline is the REST API which makes it possible to incorporate pipeline executions into shell scripts or other systems, such as web applications. Therefore, making it possible to trigger the pipeline if the system adopting it is exposed to new data or through CI tools like Jenkins and Github Actions. Other features include authentication for security and caching for skipping computations that were completed in a previous pipeline run [5].

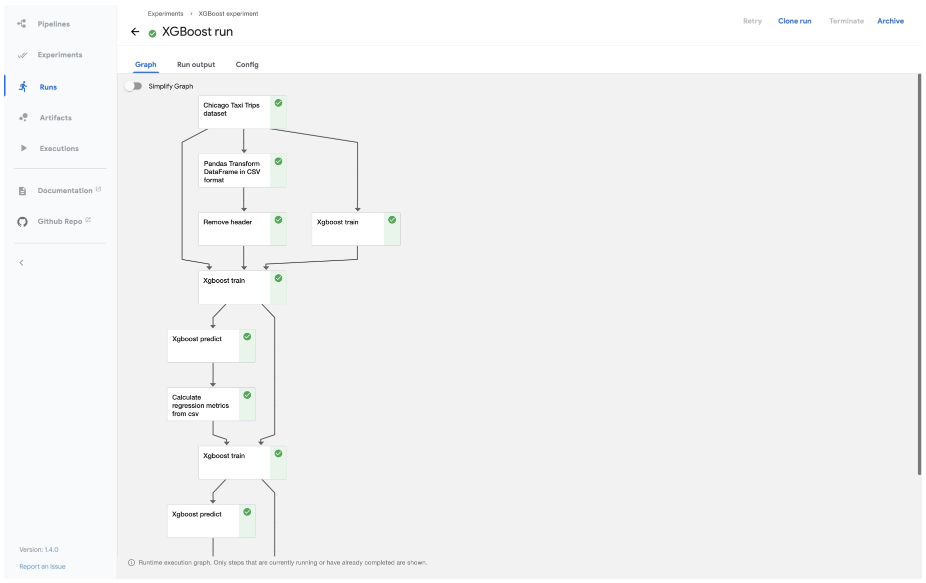


Figure 2: Kubeflow GUI of a run of a Kubeflow pipeline.

Katib is another component that allows hyper-parameter tuning, neural architecture search and early stopping for machine learning apps written in any language such as TensorFlow, PyTorch and XGBoost. It is a component for running model experiments [5]. Training operators is a component for training machine learning models in Kubeflow. It also supports multiple frameworks such as the earlier mentioned.

Finally, Kubeflow allows for integration with cloud vendors such as AWS, GCP, Azure and also supports external add-ons. An important add-on is KServe which is a serverless solution for common frameworks like TensorFlow, XGBoost, scikit-learn and PyTorch [5]. More specifically, it is possible for the user's machine learning model to use GPU autoscaling, user defined health-checks and canary rollouts for the machine learning deployments.

4 Comparison

The comparison will be focused on the two tools along with five common MLOps phases: Training, Tracking, Collaboration and Deployment.

4.1 Training

In terms of training, both Kubeflow and MLflow offer functionality for model training. In Kubeflow, this is being done inside the pipeline as an independent component. That is, the part of the user's notebook which concerns model training will be its own component on the Kubeflow pipeline. Kubeflow's component training operator also allows for distributed machine learning model training as it incorporates popular frameworks solutions for parallellization. Kubeflow allows users to split their training workflows into multiple stages which offers several benefits such as caching and reusability. This fine grained control of components is something that MLflow does not offer, as it considers the training scripts as black boxes.

4.2 Tracking

Kubeflow's tracking mechanism behaves similarly to MLflow in terms of tracking values. In Kubeflow, an experiment is created which is basically a namespace containing multiple runs of one or several pipelines. A Kubeflow pipeline has an input and an output. The outputs are called artifacts and are logged in the Kubeflow GUI whenever a pipeline is finished. Furthermore, each run will be depicted in a graph containing the flow of the pipeline. This is not provided by MLflow since there are no pipeline components provided. Rather, you input the machine learning model into the tool and MLflow will provide visualizations of tracking metrics. Hence, the tools are similar in terms of what tracking input and outputs they accept, but differ in the way of what the output looks like.

4.3 Collaboration

Both tools make it easier for users to collaborate when working on an machine learning project. For instance, in MLflow, parameters and metrics for different runs can be tracked and stored in a remote server which is accessible to all of the team through a convenient GUI. The backend database of the server can also be decoupled so it can be more scalable. In Kubeflow this is also supported by logging the artifacts of pipeline runs in a shared metadata server.

Kubeflow's has some extra collaboration features that are quite different. Kubeflow comes with native integration with notebook tools like Jupyter and provides functionality for hosting them on Kubernetes and sharing them across a team by using Access Control. MLflow in contrast does not base its collaboration functionalities on notebooks, but rather more general python projects called MLflow projects that have specific entry points (scripts) and a specific conda environment.yaml file.

Also relevant to collaboration is versioning of the models. As mentioned, MLflow offers the MLflow registry component which offers versioning of the models as well as annotating them. Kubeflow however does not offer such functionality and many users are using MLflow within their Kubeflow projects specifically for the model registry [4].

4.4 Deployment (for production)

Both MLflow and Kubeflow allow for standalone deployments. That is, machine learning model deployment without third party cloud deployment such as AWS or Azure. However, the standalone deployments differ. For MLflow, machine learning model deployments are usually done by creating docker images with the V2 specification. Alternatively, MLflow can deploy trained python_function flavours of a model to Amazon SageMaker and Microsoft Azure machine learning with built in integrations.

On the contrary, Kubeflow does standalone deployments of the entire or parts of the pipeline using Kubernetes clusters, which the user needs to set up along with kubectl - a CLI tool for interaction with the cluster. Similarly, MLflow currently has an experimental feature for deploying a machine learning model to a Kubernetes cluster, but only through the MLflow projects. If the user rather chooses a third party cloud vendor for deployment, Kubeflow integrates with virtually any cloud provider that offers a managed Kubernetes service.

5 Reflection

Overall Kubeflow seems more suitable for large projects. Maintaining a Kubeflow cluster introduces a lot of technical work within the team and although Kubeflows functionality is extensible it takes quite a bit of time to set up.

On the contrary, for smaller projects MLflow, offers easy to set up solutions for small teams with minimal boilerplate. For example, researchers can write code in their favourite framework and with only one line of code perform tracking without the need to define complex pipelines. [3]

6 Conclusion

Kubeflow and MLflow are two MLOps tools that emphasize automatization of a machine learning model's life cycle. This for facilitating the machine learning model's workflow. Their main components and features have been described and compared. The findings indicate that components which are similar essentially offer the same kind of functionality, although there are a few differences in how it is being executed (e.g. Docker vs Kubernetes) and visualized (e.g. graph of pipeline in Kubeflow) in the Graphical User Interface. However, Kubeflow offers more functionality compared to MLflow since it offers an end-to-end solution with the exception of the model registry which is unique to MLflow.

References

[1] Ekaba Bisong. "Kubeflow and kubeflow pipelines". In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Springer, 2019, pp. 671–685.

[2] L Columbus. *The state of MLOps in 2021 is dominated by startups.* `https://venturebeat.com/2021/09/22/the-state-of-mlops-in-2021-is-dominated-by-startups/`. Accessed: 2022-05-16.

[3] Christof Ebert et al. "DevOps". In: *Ieee Software* 33.3 (2016), pp. 94–100.

[4] *GitHub issue on Kubeflow's repository regarding lack of model registry.* `https://github.com/kubeflow/kubeflow/issues/5093`. Accessed: 2022-05-16.

[5] *Kubeflow documentation.* `https://www.Kubeflow.org/docs/`. Accessed: 2022-05-16.

[6] et al. Medel Víctor. *Characterising resource management performance in Kubernetes.* `https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/docs/vmware-kubernetes-up-running-dive-into-the-future-of-infrastructure.pdf`. Accessed: 2022-05-16.

[7] *MLflow documentation.* `https://www.mlflow.org/docs/latest/index.html`. Accessed: 2022-05-16.

[8] Philipp Ruf et al. "Demystifying MLOps and Presenting a Recipe for the Selection of Open-Source Tools". In: *Applied Sciences* 11.19 (2021), p. 8861.

[9] Damian A Tamburri. "Sustainable mlops: Trends and challenges". In: *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE. 2020, pp. 17–23.

[10] Mark Treveil et al. *Introducing MLOps.* O'Reilly Media, 2020.