Assignment 2

Cole Lewis, Will Duggan, Daniella Mallari

Introduction

Kubeflow is an cloud-native, open-source, machine learning platform built for use with Kubernetes, a platform for deploying and managing container runtimes such as Docker, containerd, cri-o, and others. Kubeflow, with its integration into Kubernetes, can quickly and reliably provide a means of building machine learning systems and pipelines from development to production either on containers or across distributed systems. Kubeflow collectively is made of five software components that each serve a different aspect of machine learning pipeline/model creation and subsequent deployment for production. These components are: Kubeflow Notebooks, Kubeflow Pipelines, Kubeflow Training Operator, KServe, and Katib. Through a combination of these and other components chosen by the user, Kubeflow serves as a robust means of model creation and deployment through the entire process.

“Nature of the Machine Learning Task”

As a virtue of being a platform for construction and deployment of machine learning models and their pipelines, Kubeflow is inherently well-suited to accommodate any sort of machine learning task it is given, provided that the code and underlying frameworks are fit to carry them through. Kubeflow was originally developed by Google and began as an open-sourcing of Google’s own internal use of TensorFlow, which initially was based on a pipelining system named TensorFlow Extended meant to be a simple way of propagating TensorFlow jobs using Kubernetes. Due to being cloud-native by design, Kubeflow is best used for tasks that employ cloud resources such as data storage, virtual machines, distributed computing, and other remote resources. This is especially useful in a research capacity in which large datasets, scattered teams of people, continuity, and scalability are highly supported.

“Modeling and Experimentation Ideas”

To create ideas for experimentation, Kubeflow can perform a Neural Architecture Search (NAS) on a set of given parameters. Once the parameters are specified, the objective metric to be optimized also needs to be specified. Examples of this metric could be accuracy, loss, etc. Any model can be used with Kubeflow, and experiments can be tracked easily and conveniently. Kubeflow experiments track all parameters specified for the run, so desired results are easily accessible and the experiments are reproducible. With the hyperparameter tuning feature, an objective metric can be optimized to ensure the best experiment for a given model and architecture.

“Transition from Development to Production”

Upon research of notable uses, our team encountered examples of Kubeflow implementations that span the entire process and found that in practice, usage patterns arose pertaining to how each component is used and leveraged with popular machine-learning frameworks to develop, monitor, and deploy the implementation in question.

The vast majority of implementations begin with deployment of a Kubeflow cluster, either on a local machine or using Google Cloud AI. It is very common to find other frameworks and deployment infrastructures made by Google used in conjunction with Kubeflow, this can be considered a key benefit due to how common these tools already are in machine learning development (i.e., TensorFlow, Google Cloud AI, Google Kubernetes Engine, etc.)

Regardless of the exact containerization platform used (Docker, Kubernetes, containerd, etc.), Kubeflow by design necessitates use of a cluster, or container, to hold and execute pipelines as well as model training. All researched Kubeflow uses have done containerization via Docker or Kubernetes, although alternatives are easily supported. This approach confers a major advantage in being reproducible. Much like how Python’s `venv` module allows an application or project’s libraries and dependencies to be portable and easily bundled, containers provide a method of identically duplicating results. In use with machine learning, this means bundling data preprocessing, model training, and model deployment into a single package that can be run while abstracting away differences between machines such as operating systems and infrastructure.

Kubeflow Pipelines collect each component of the workflow and how they work together. This combination is succinctly represented in graph form and stored as an image on your container of choice. Kubeflow offers a native user interface via Kubeflow Pipelines UI that allow the user to modify, schedule, and execute created pipelines. The included Kubeflow Pipelines SDK offers a set of Python packages for use in compiling to workflow YAML specifications, interacting with pipeline components and interfacing with the Kubeflow Pipelines API

Kubeflow Notebooks provides a means of using web-based development environments from Jupyter Notebook or other alternatives within the container or Kubernetes cluster. Within the observed implementations, Kubeflow Notebooks were used to centralize code and documentation together to do tasks such as train the model while also offering a convenient form of documentation alongside it, explaining what each block of code is meant to do. Besides training, these notebooks would also be used to coordinate pipeline creation, preprocess data, push deployments, and other various tasks all from within the container or cluster at hand.

Once the container or cluster is initialized and the pipeline is built, the model is ready to be trained. We found that many end-to-end Kubeflow implementations, by virtue of using TensorFlow as a framework, mostly opted to use TFJob to run TensorFlow training jobs. TFJob is one of multiple custom resources, or extensions to the Kubernetes API, that constitute the full set of Kubeflow Training Operators. TFJob is a YAML resource that lays out permissions, images, commands, volume mount paths, and API credentials for pods running TensorFlow.

Alternatively, the Kubeflow Pipelines SDK allows for a pipeline to be compiled in its entirety, including model training, which is framework-agnostic mode. In place of a TFJob resource, the specifications of how the training job, or jobs, are to be run from a notebook or other program within the container or node being used.

After the model is trained to satisfaction comes the question of serving and deploying the model. While Kubeflow has a serving component, KServe, in its stack, the implementations researched either used Google Cloud AI or Seldon Core to deploy the model and make inferences. Yet again, the portability of containerization allows them to be deployed easily on platforms such as Google's Docker Registry if using Docker, or Seldon, which is Kubernetes-native and deploys Kubernetes pods.

"Validation Datasets"

Kubeflow pipelines are a portable, scalable, and reproducible way to run different steps in the ML system pipeline. One such component in the ML system pipeline is the loading and transforming of the training and validation sets. Components can be built modularly, allowing reuse of components such as loading different datasets. Kubeflow's reusable components allow different ML Engineers across a team to test their system with different validation sets, giving a more holistic evaluation of the ML system. This allows for the validation set to easily be changed between runs. Kubeflow's MLServing allows for automatic monitoring to detect things like data drift (more on that in the next section), which can be a signal to the ML Engineer that the validation set needs to be adjusted to the same distribution as the new input data.

"Monitoring"

For monitoring and managing the lifecycle of applications deployed on Kubernetes and Openshift, there is Kubeflow Operator. The Operator is an open source toolkit and includes tools for operators such as testing, building, and managing. In the case of monitoring, the Operator has a tool specifically for storing data in a database and allows the cataloged data to be accessed with the Operator Lifecycle Manager. The Operator Lifecycle Manager is then able to install, upgrade, and access other control operators, aptly named the "operator of operators". With this toolset, the operator is able to be flexible enough to allow the user to customize components, and provide easy ways to upgrade and update Kubeflow applications when needed.

Kubeflow's ML Serving allows a user to construct a graph consisting of thingslike data processing, post-processing, model ensembles, and model servers. ML Serving also provides native model monitoring, with features such as auto drift detection, custom metrics, and a custom plug-in for Grafana.

Stackdriver is built directly into Kubeflow and allows for different Logging and Monitoring functions. Stackdriver's functions are specifically focused on the resource usage of Kubeflow (i.e. CPU/Memory usage). It's important to monitor resource usages to ensure that a ML system is running without high utilizations that could result in a slowdown. Stackdriver's monitoring is compatible with the Prometheus data model, which is a commonly used tool for monitoring Kubernetes workflows.