Deployment of ML Models using Kubeflow on Different Cloud Providers

Aditya Pandey (ap6624)
Maitreya Sonawane (mss9240)
Sumit Mamtani (sm9669)

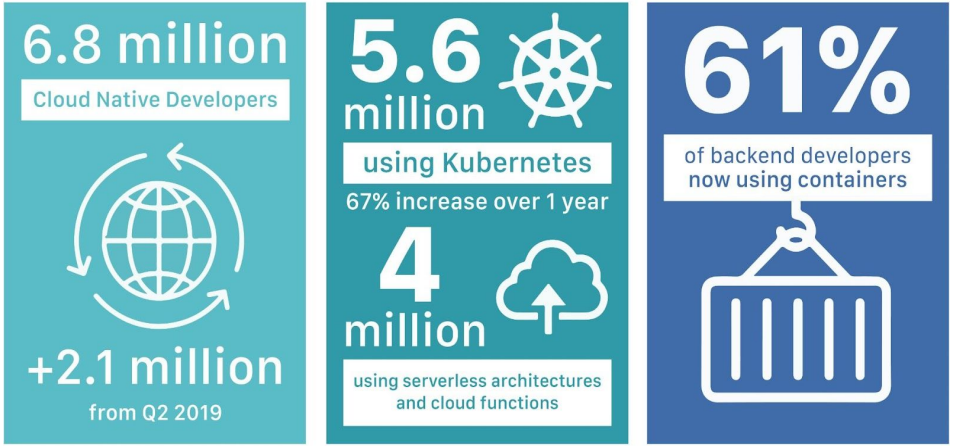
CSCI-GA 3033: Cloud and Machine Learning

Content

- Background & Motivation
- Related Work
- Kubeflow
  - Architecture
  - Setup
  - Pipelines
  - Predictions
- Results & Comparisons
- Insights
- Challenges
- Discussion & Future Work

Background & Motivation

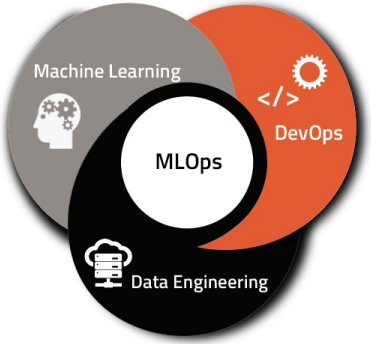
- Increased use of Cloud and Distributed Systems and Kubernetes is the #1 choice for most companies.
- Rise of container usage on distributed systems and edge computing.



https://www.cncf.io/blog/2021/12/20/new-slashdata-report-5-6-million-developers-use-kubernetes-an-increase-of-67-over-one-year/

- New fads such as Machine Learning, AI and Deep Learning are great when working in a developmental environment - Translation to production?
- Hidden Technical Debt in Machine Learning Systems - While using ML to build complex prediction systems seems easy, there is a heavy cost of *maintenance*. [Paper]

Background & Motivation



https://www.analyticsvidhya.com/blog/2021/06/mlops-operationalizing-machine-learning-models-in-production

- Productivity: Self-service environments for data engineers and data scientists
- Repeatability: Automating all the steps helps you ensure a repeatable process, including how the model is trained, evaluated, versioned, and deployed.
- Reliability: Incorporating CI/CD practices allows quick deployment with increased quality and consistency.

Inefficient tools and infrastructure

Lack of iterative deployment

Need of Automated CI/CD pipelines

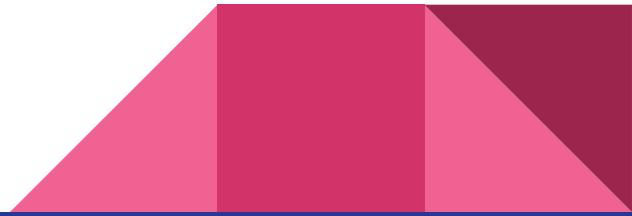
Data growth = More computing power

Related Work

- MLflow
  - Created by Databricks
  - An open source platform for managing the end-to-end machine learning lifecycle
  - Python Library - Useful for tracking ML code
- Airflow/Argo
  - Both tools are general task orchestration platforms
  - Part of Kubeflow is actually build on Argo (which also runs on Kubernetes)

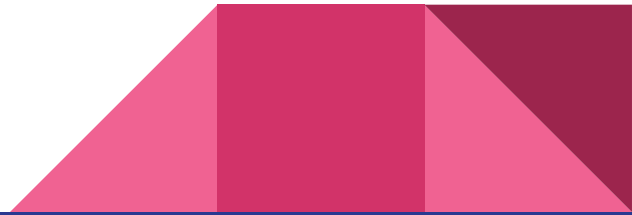
* Airflow and MLFlow do not run on Kubernetes

- Kubeflow combines the best of ML Flow and uses a DAG based approach to run on Kubernetes

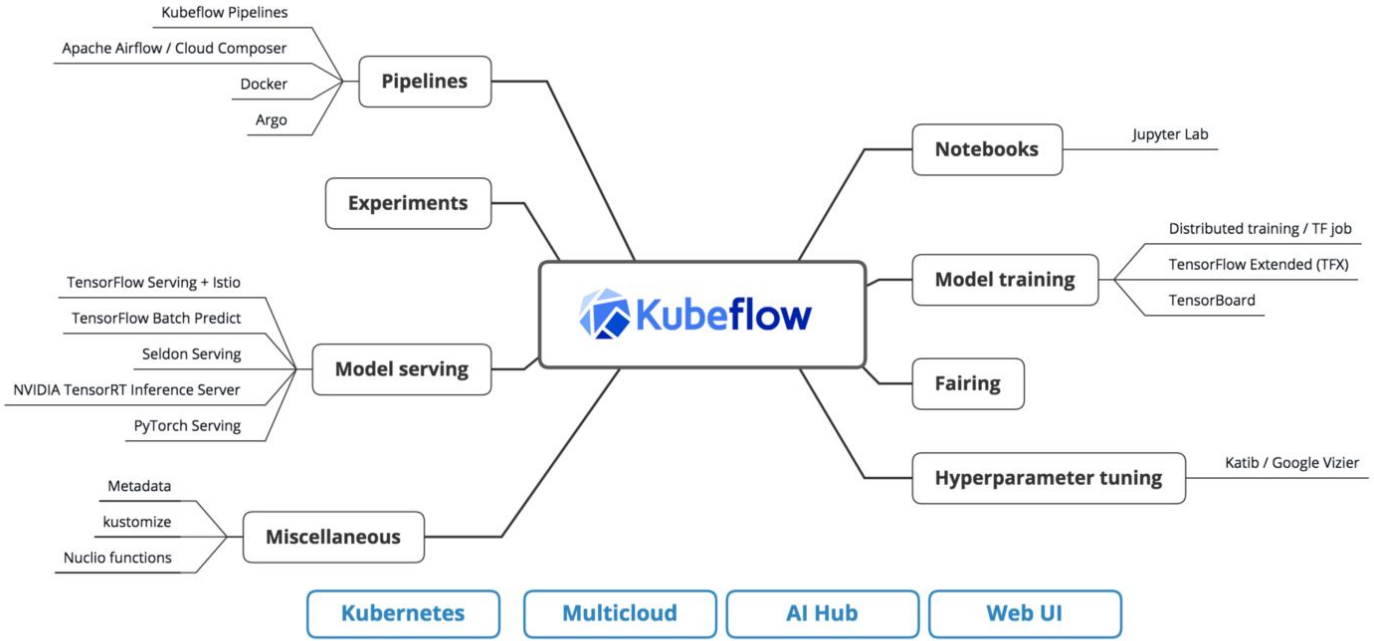


What is Kubeflow?

- Kubeflow - Initially a project at Google to run TensorFlow jobs on Kubernetes
- It is now a complete ML toolkit -
  - An open source platform that allows Machine Learning pipelines to run on Kubernetes Clusters
  - Essentially an end-to-end ML Stack orchestration toolkit
- Open source - Not locked in to any Cloud provider
- Abstracts most Kubernetes concepts to let ML Developers and Engineers



Kubeflow Components + Architecture



https://medium.com/@michal.brys/kubeflow-a-machine-learning-toolkit-for-kubernetes-d8686f6c91b6

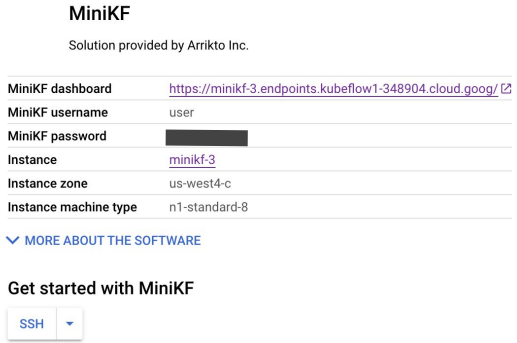
Baselines

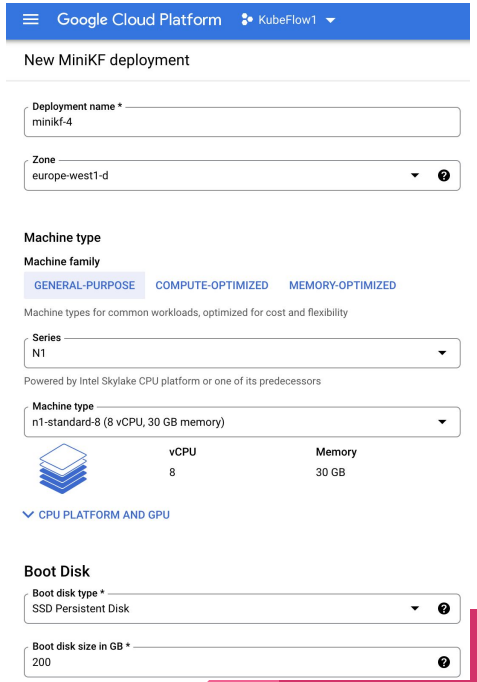
- To compare the ease-of-use and performance aspects of Kubeflow, we run our training jobs on 2 types of baseline platforms -

NYU Greene Cluster + Server	Basic Kubernetes (on IBM Cloud)
Performed MNIST training on NYU Greene Cluster and inference hosting on linserv machine.	Performed MNIST training and MNIST inference hosting in the IBM Kubernetes cluster.
All environment setup/resource requesting needs to be done manually	Environment and resource components are handled by Kubernetes
No docker/k8s support	Supports generic containers

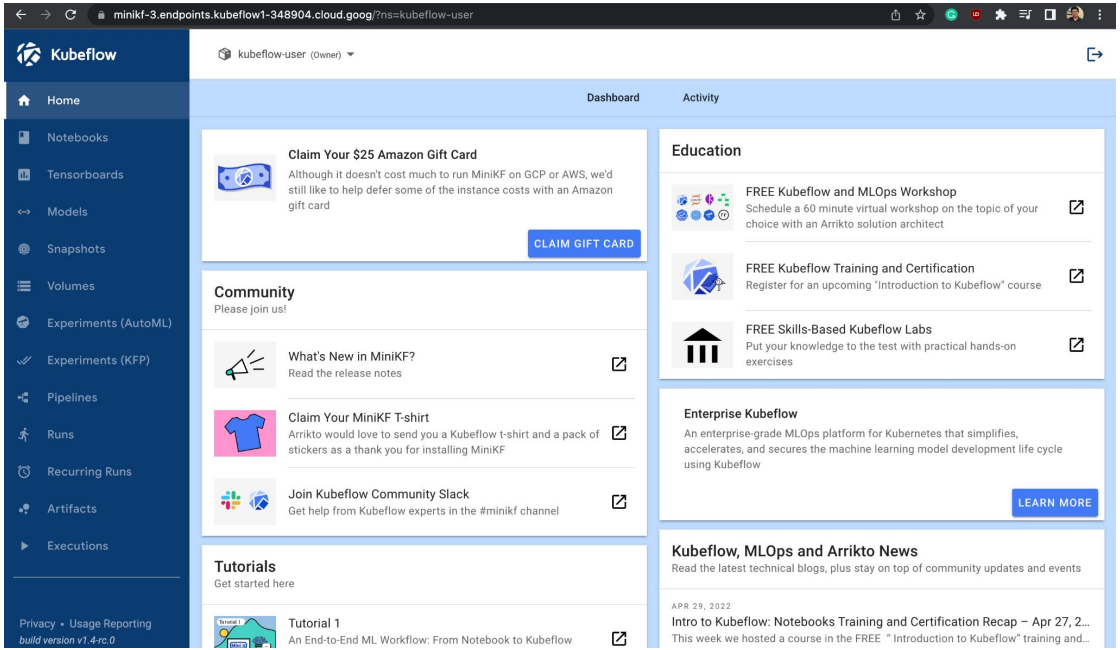
Setup - Google Cloud Platform (GCP)

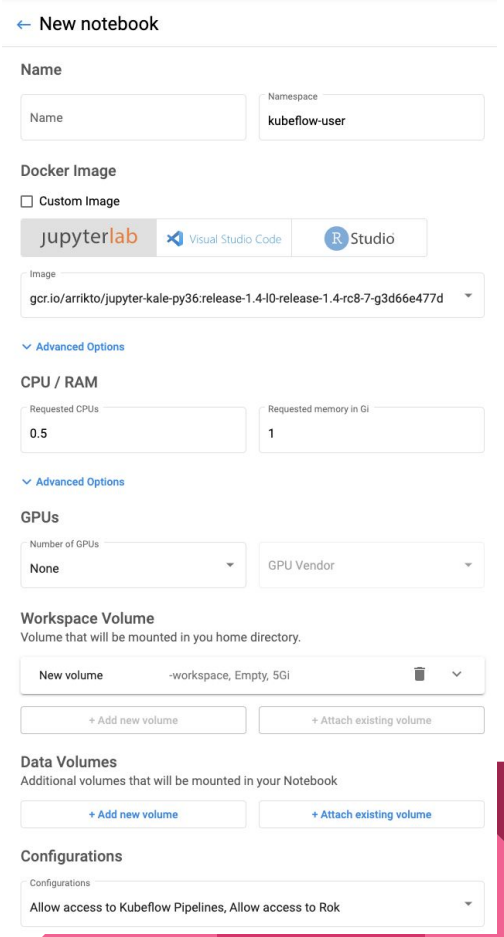
- MiniKF - single-node, full-fledged Kubeflow deployment
  - Create a project on GCP, ensure billing is enabled and IAM roles include editor privileges
  - Launch MiniKF from Marketplace, define VM resources in ‘Configure and Deploy’
    [8 vCPU, 30 GB Memory, 200 GB SSD Boot Disk, 500GB Standard Data Disk]
  - SSH button to install minikf, gives URL for accessing Kubeflow Dashboard





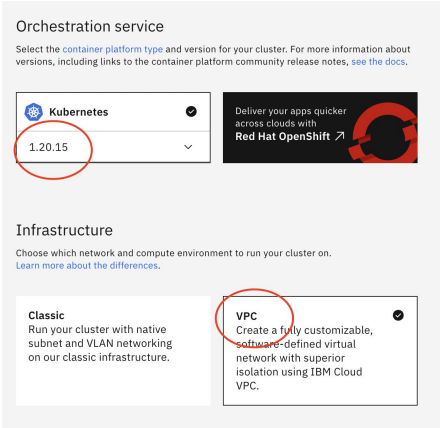
Setup - Google Cloud Platform (GCP)





Setup - IBM Cloud

- Virtual Private Cloud + Block Storage -
  - To create an effective Kubeflow platform, we create a VPC and set up/enable subnets, block storage and routing tables under the same region
- Kubernetes Cluster -
  - Choose the VPC Option



Prerequisites

- `Kubernetes` (up to `1.21`) with a default StorageClass
  - ⚠️ Kubeflow 1.5.0 is not compatible with version 1.22 and onwards. You can track the remaining work for K8s 1.22 support in kubeflow/kubeflow#6353
- `kustomize` (version `3.2.0`) (download link)
  - ⚠️ Kubeflow 1.5.0 is not compatible with the latest versions of of kustomize 4.x. This is due to changes in the order resources are sorted and printed. Please see kubernetes-sigs/kustomize#3794 and kubeflow/manifests#1797. We know this is not ideal and are working with the upstream kustomize team to add support for the latest versions of kustomize as soon as we can.
- `kubectl`

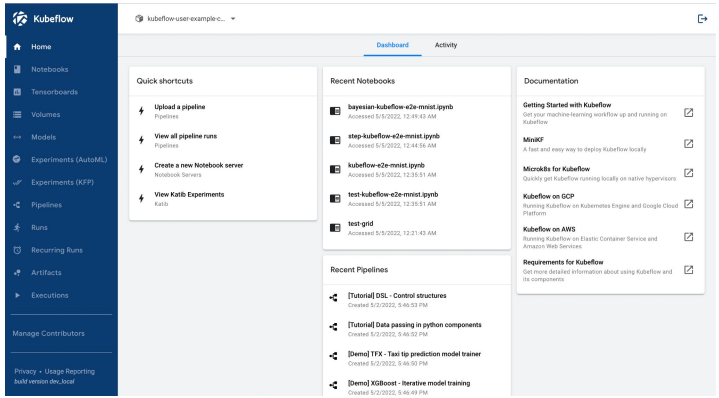
- Install kustomize (on your machine with IBM Cloud CLI) and make sure kubectl commands work
- *Note: The versions are **important** as the next installation steps fail*

Setup - IBM Cloud

- `git clone https://github.com/IBM/manifests.git && cd manifests`
- `while ! kustomize build example | kubectl apply -f -; do echo "Retrying to apply resources"; sleep 10; done`

The two commands above together will apply most of the configurations needed to install a basic flavour of Kubeflow

- To view the Kubeflow dashboard, we just need to activate the ingress service using -
- `kubectl port-forward svc/istio-ingressgateway -n istio-system 8080:80`



Setup - IBM Cloud

There are a few oddities/specific points to keep in mind when integrating Kubeflow with IBM Cloud

- The UI or any service created by default is not exposed to the internet. We need to use a LoadBalancer/expose a Port for this purpose - `kubectl patch svc istio-ingressgateway -n istio-system -p '{"spec":{"type":"LoadBalancer"}}'`
- All of our deployments currently run on HTTP by default. To access Jupyter Notebooks, we need to secure our endpoints with TLS

References -

- https://www.civo.com/learn/get-up-and-running-with-kubeflow-on-civo-kubernetes#step-4-enable-https-to-access-kubeflow
- https://www.kubeflow.org/docs/distributions/ibm/deploy/authentication/

Code Approach vs E2E Approach

We have compared 2 approaches to running an ML pipeline on Kubeflow -

1. Running a container image directly via a TFJob (End to End).

```
},
"spec": {
    "containers": [
        {
            "name": "tensorflow",
            "image": "docker.io/liuhougangxa/tf-estimator-mnist",
            "command": [
                "sh",
                "-c"
            ],
            "args": [
                "python /opt/model.py --tf-export-dir=/mnt/export
            ],
            "volumeMounts": [
                {
                    "mountPath": "/mnt/export",
                    "name": "model-volume"
                }
            ]
        }
    ],
```

2. Creating a kubeflow pipeline by writing our own TF code over a base image.

```
# create light weight components
download_op = comp.func_to_container_op(download_data,base_image="python:3.7.1")
load_op = comp.func_to_container_op(load_data,base_image="python:3.7.1")
preprocess_op = comp.func_to_container_op(preprocess_data,base_image="python:3.7
modeling_op = comp.func_to_container_op(modeling, base_image="tensorflow/tensorf
predict_op = comp.func_to_container_op(prediction, base_image="tensorflow/tensor
```

Create kubeflow pipeline components from images

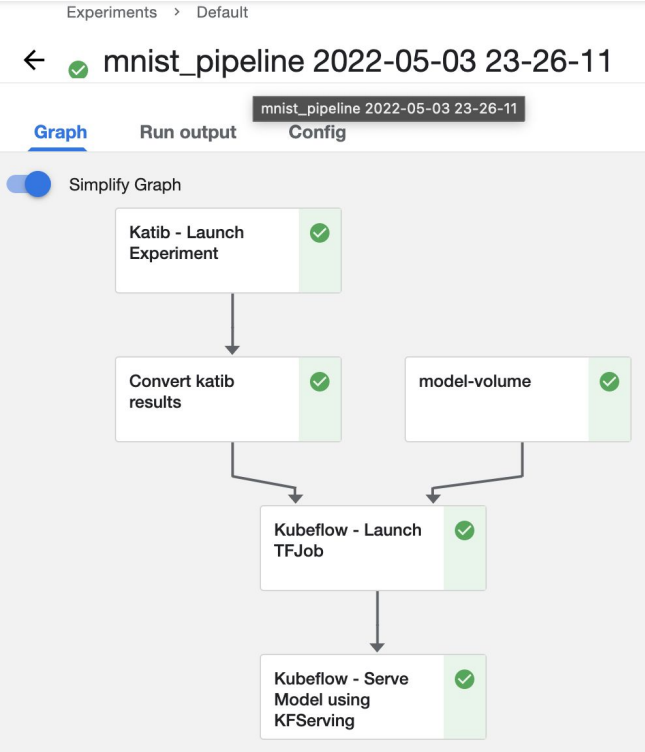
```
#initializing the classifier model with its input, hidden and outpu
hidden_dim1=56
hidden_dim2=100
DROPOUT=0.5
model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(filters = hidden_dim1, kernel_size =
                  activation ='relu'),
        tf.keras.layers.Dropout(DROPOUT),
        tf.keras.layers.Conv2D(filters = hidden_dim2, kernel_size =
                  activation ='relu'),
        tf.keras.layers.Dropout(DROPOUT),
        tf.keras.layers.Conv2D(filters = hidden_dim2, kernel_size =
                  activation ='relu'),
        tf.keras.layers.Dropout(DROPOUT),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(10, activation = "softmax")
        ])

model.build(input_shape=(None,28,28,1))
```

Pipeline

- Katib's Hyperparameter tuning task
  - Objective set to minimize loss with goal = 0.001, random search over hyperparameters: '*learning_rate*' [0.01-0.05] and '*batch_size*' [80-100]
- TFJob training task
  - Use the best hyperparameters found from Katib's experiment to train same model - LeNet, an image classification model using the MNIST dataset
- KServe Inference
  - Create a serving component URL that will be used in inference of the model
- At last run the Kubeflow Pipeline with end to end MNIST model with hyperparameter tuning, training and inference, we create the Volume to train and serve our model here and run Kubeflow pipeline using the same namespace as the user

Pipeline



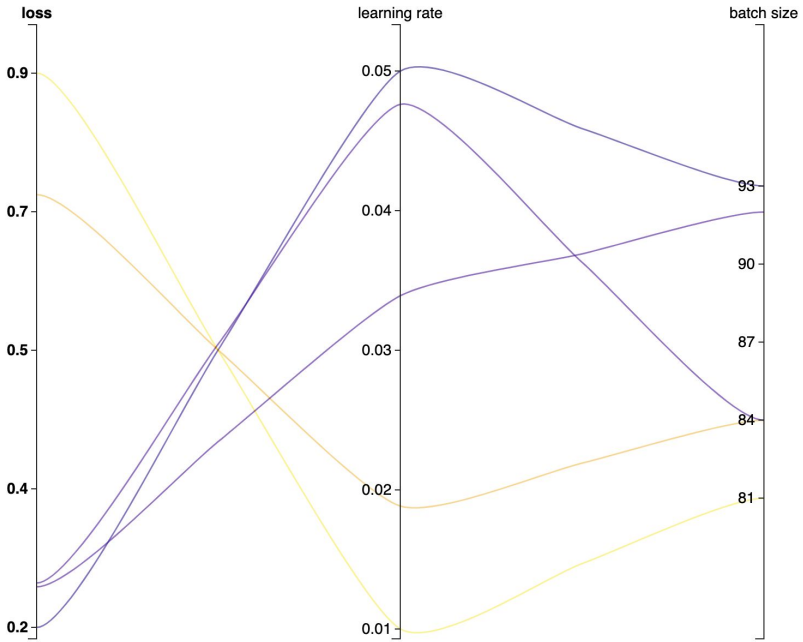
- For tuning, we used **AutoML** in the form of **Katib** integration with **Kubeflow.**
- We used random, bayesian and step algorithms.

Based on all the trials we did, we obtained the following results -

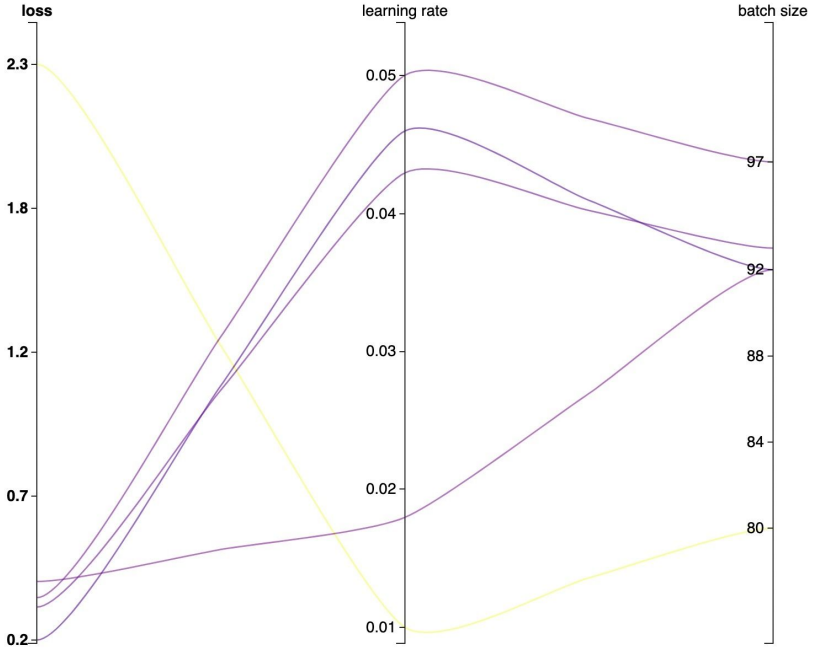
Metric	Best Trial Performance Loss	Tuned Learning Rate	Tuned Batch Size
IBM	0.1876	0.453	92
GCP	0.2047	0.498	93

Pipeline

GCP



IBM Cloud



Prediction

```
image_url = "https://i.imgur.com/6qsCz2W.png"
image = Image.open(requests.get(image_url, stream=True).raw)
data = np.array(image.convert('L').resize((28, 28))).astype(np.float).reshape(-1, 28, 28, 1)
data_formatted = np.array2string(data, separator=",", formatter={"float": lambda x: "%.1f" % x})
json_request = '{{ "instances" : {} }}'.format(data_formatted)

url = "http://mnist-e2e-4-predictor-default.kubeflow-user.svc.cluster.local/v1/models/mnist-e2e-4:predict"
response = requests.post(url, data=json_request)
print("Prediction for the image")
print(response.json())
display(image)
```

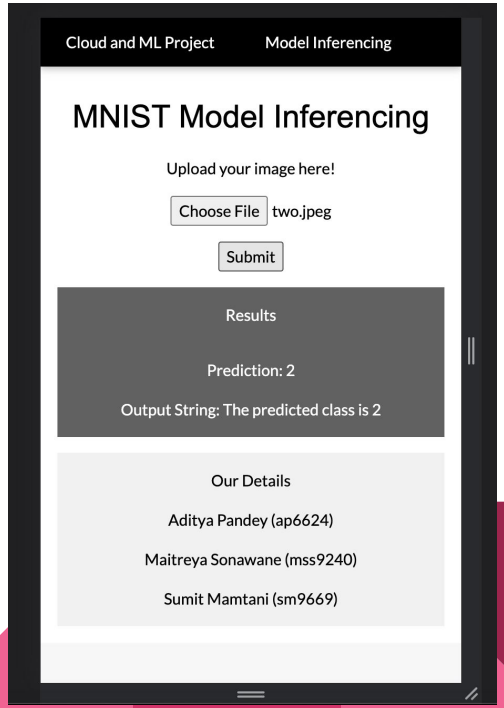
```
Prediction for the image
{'predictions': [{'predictions': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0], 'classes': 7}]}
```

With Kubeflow (GCP and IBM Cloud)

```
maitreya@10-16-62-118 dataset % curl -X POST -F image=@zero.jpg 'http://52.118.148.144:5000/predict'
{"result": 0, "status": "Prediction Complete"}%
maitreya@10-16-62-118 dataset %
```

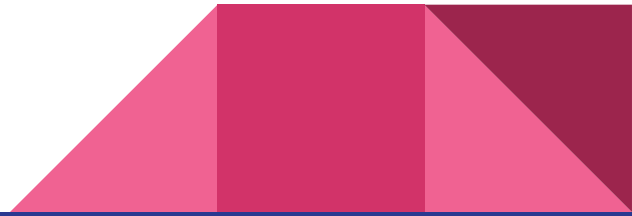
Without Kubeflow

UI on K8



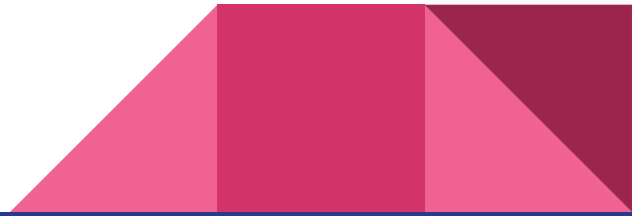
Kubeflow Addons Used

- Istio on Kubeflow -
  - Istio is an open source framework used by Kubeflow to enable end-to-end authentication and access control.
  - It is basically a service mesh for different microservices to communicate with each other. - It manages most of the ingress points and we can expose endpoints using this.
- KServe -
  - KServe (formerly KFServing) is what we use to provide an inferencing service for our Kubeflow pipeline.
  - It provides a standard interface to use our model for predictions.
  - Seldon Core is an alternative to this.

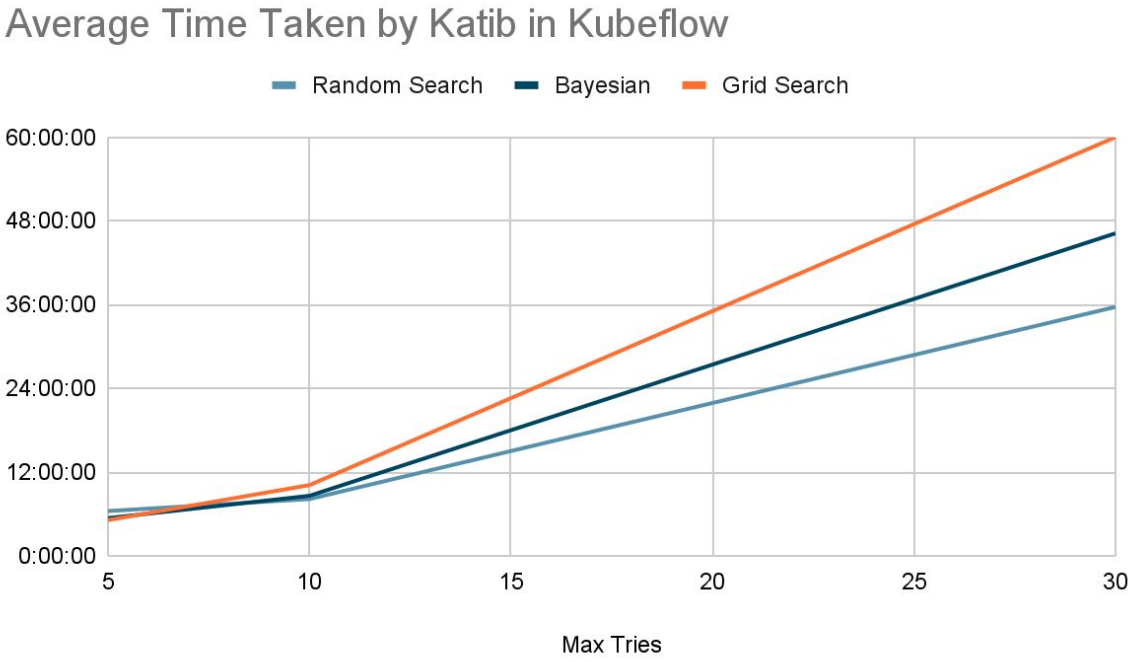


Different Experiments Run

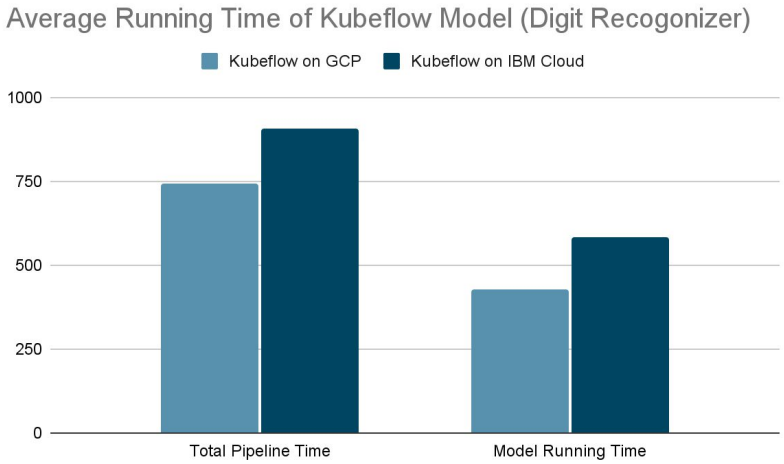
- Running MNIST on NYU Greene Cluster + Hosting inference on linserv
- Running a Basic MNIST Image on Kubernetes (on IBM Cloud)
- Running MNIST on Kubeflow in IBM Cloud - Using E2E and Code Approach
- Running MNIST on Kubeflow in Google Cloud (GCP) - Using E2E and Code Approach

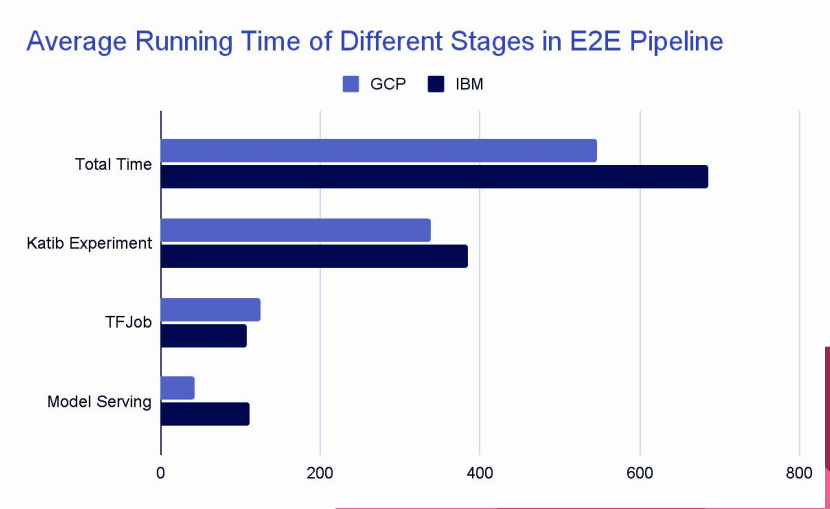


Results & Comparisons

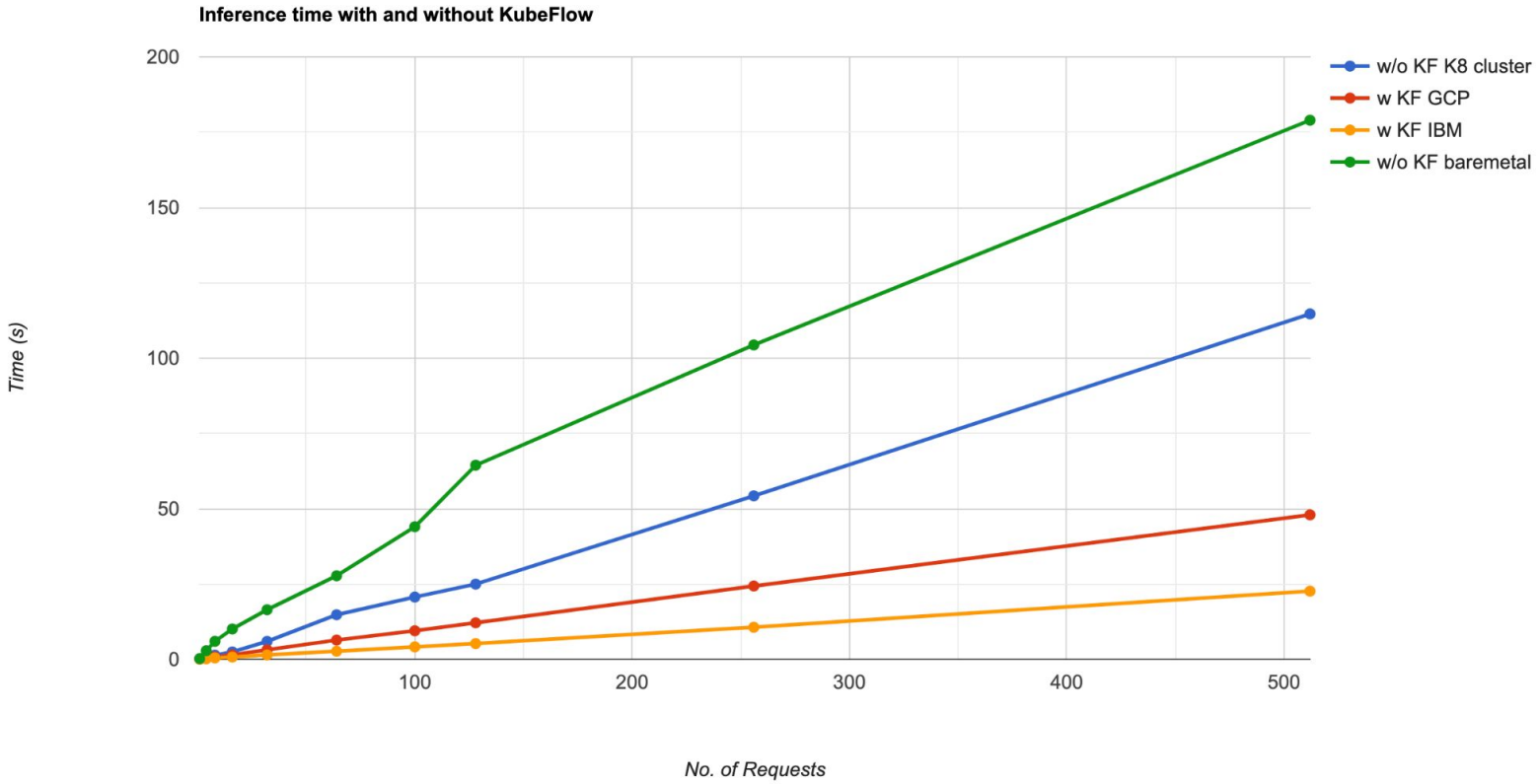


Results & Comparisons



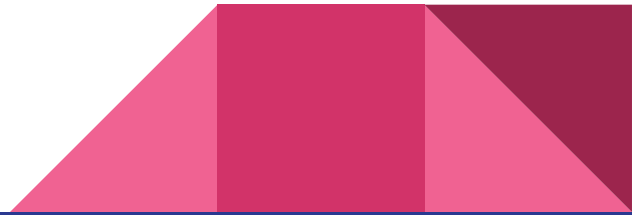


Results & Comparisons



Insights

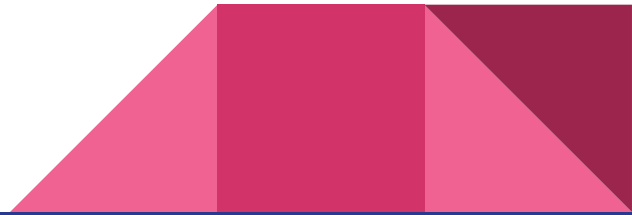
- MNIST code running on the Kubeflow on IBM cloud has the least inference time among all the models
  - To test the inference performance, we performed a sort of stress test on the inference endpoint for both clouds and noted the total response time.
  - One reason we think that IBM Cloud has a lower inference time is because all our K8s components on IBM Cloud are defined inside the same VPC in the same region.
- The Duration for running E2E pipeline is less for Kubeflow on GCP
  - We feel that the total duration for running the pipeline on GCP is lower as the cluster is more powerful and the contention of resources is lower.



Insights

- The overall process of creating a cluster and using Kubeflow on it was easier on GCP for a few reasons - easier availability of documentation, automatic HTTPS endpoint securing, easy access to KF pipelines from Notebooks, etc.

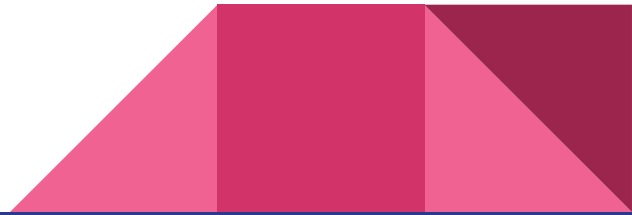
- While IBM cloud has all these features, it is not as intuitive to enable them



Insights

While Kubeflow has a lot of advantages, especially during Model training and inferencing, there are a few pitfalls that we feel prevent a more widespread adoption of the framework.

- The difficulty with the initial installation and authentication setup makes it painful to even start with KF.
- As Kubeflow uses different versions of different components (such as Istio), upgrading individual components is a risky task.
- Out of date documentation + Broken Links.



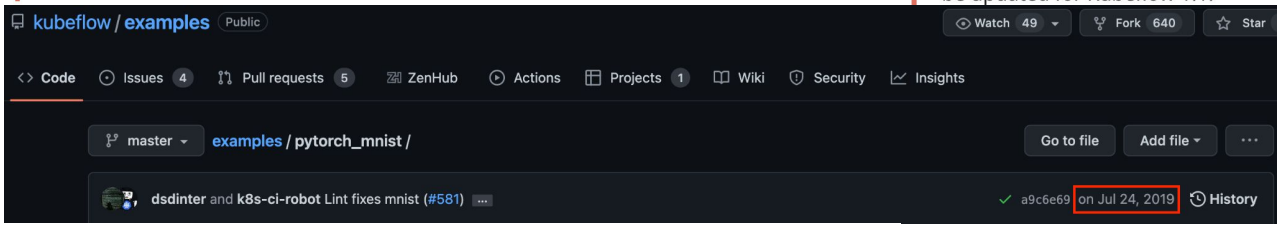
Challenges

IBM Cloud Kubernetes Versions	Kubeflow 1.5.0
1.20	**Compatible**
1.21	**Compatible**
1.22	**Incompatible**

Deploy using UI

Instructions for using the UI to deploy Kubeflow on Google Cloud Platform (GCP)

> **No longer supported**
>
> Starting with Kubeflow v1.1.0 deploying Kubeflow via the click to deploy web application is no longer supported. Please use kustomize and kpt to deploy Kubeflow.



- **Q:** What versions of Istio, Knative, Cert-Manager, Argo, ... are compatible with Kubeflow 1.4?
  A: Please refer to each individual component's documentation for a dependency compatibility range. For Istio, Knative, Dex, Cert-Manager and OIDC-AuthService, the versions in `common` are the ones we have validated.
- **Q:** Can I use the latest Kustomize version (`v4.x`)?
  A: Kubeflow 1.4.0 is not compatible with the latest versions of of kustomize 4.x. This is due to changes in the order resources are sorted and printed. Please see kubernetes-sigs/kustomize#3794 and kubeflow/manifests#1797. We know this is not ideal and are working with the upstream kustomize team to add support for the latest versions of kustomize as soon as we can.

Component Specification

Definition of a Kubeflow Pipelines component

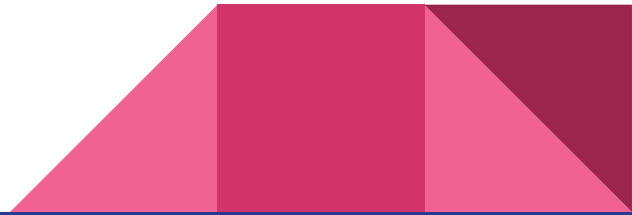
> **Out of date**
>
> This guide contains outdated information pertaining to Kubeflow 1.0. This guide needs to be updated for Kubeflow 1.1.

KFServing docs are impossible to follow #2589

Closed **ulrikpl** opened this issue on Apr 8, 2021 · 9 comments

Discussion

- In this project, we presented a deep dive into integrating Kubeflow on both IBM Cloud and GCP, while comparing them with similar models deployed on K8 cluster/trained on NYU HPC w/o Kubeflow
- We found that while duration of E2E run for Kubeflow on GCP was the least, IBM Cloud surpassed every other model to give fastest inference time.
- With better documentation and community support, Kubeflow has the right tools to be a successful framework.
- While Kubeflow is great for running ML Jobs on Kubernetes; environments such as HPCs and Big Data Systems have their own flavours of MLOps



Future Work

- Explore the integration of Kubeflow with GPU enabled Clusters and Notebooks.
- Increase the depth of the neural network to see even more improvement over bare metal.
- Increase number of trials in terms of Katib hyper-parameter tuning as well as different model architectures.
- Explore compatibility of different frameworks such as PyTorch with Kubeflow.
- Explore an ML problem from a different domain (such as Speech/Text)
- Become an open-source contributor - https://v1-5-branch.kubeflow.org/docs/about/contributing/