Comparison of Deploying Deep Learning Models with MLflow on Different Cloud Platforms

Aditya Dixit
ad5157@nyu.edu

Haseeba Rahman
hf2313@nyu.edu

Nivedita Patel
np2457@nyu.edu

December 2022

Abstract

In this project, we present a study of the deployment of a deep learning model with MLFlow on Kubernetes clusters in three different cloud platforms, namely, IBM Cloud, Google Cloud Platform and Amazon Web Services. We provide details on cloud services involved in the deployment process and compare the deployment processes on all the cloud platforms. Finally, we analyze the performance of the model in terms of training time, job execution time, test loss and data loading time, which were measured using MLFlow metrics and tracking in the different cloud platforms. We will also summarize the key findings and provide some insights into the factors that may influence the choice of cloud platform for MLflow deployment based on our analysis.

1 Introduction

Deep Learning is transforming numerous industries such as healthcare, education, transportation, among many others. However, productionizing a deep learning model can be cumbersome. Machine learning workflows encompass several technical and logistical difficulties that can be hard to foresee and resolve, such as model performance evaluation, efficiently maintaining different versions of a model, among others. Teams developing models are widely using MLOps methodologies to simplify and enhance the processed involved in managing machine learning models.

1.1 MLOps

Machine Learning Operations (MLOps) aims to streamline the machine learning lifecycle, which entails several intricate steps, including ingesting and processing sizable amounts of data, training a model, fine-tuning its hyperparameters, deploying the model into production, and then monitoring, validating, and maintaining it. The machine learning lifecycle's experimentation, iteration, and continual improvement are all included in MLOps.

Employing MLOps processes in machine learning deployment has numerous benefits:

- By enabling users to repeatedly execute the algorithm using various datasets and get the same (or comparable) results on a project, it improves reproducibility.
- We can log metrics and hyperparameters during machine learning executions and run diagnostics.
- MLOps allows users to monitor models and get a live picture of performance metrics during model training.
- Version control is essential to reproduce results of models trained with different hyperparameters, features, and other model metadata. Utilizing MLOps tools makes tracking these modifications much easier.

1.2 MLflow - A platform for the Machine Learning Lifecycle

A number of tools are available for creating MLOps workflows. Some of the popular platforms are MLflow [10], Kubeflow [9] and Data Version Control (DVC) [3]. However, in our project, we explore MLflow.

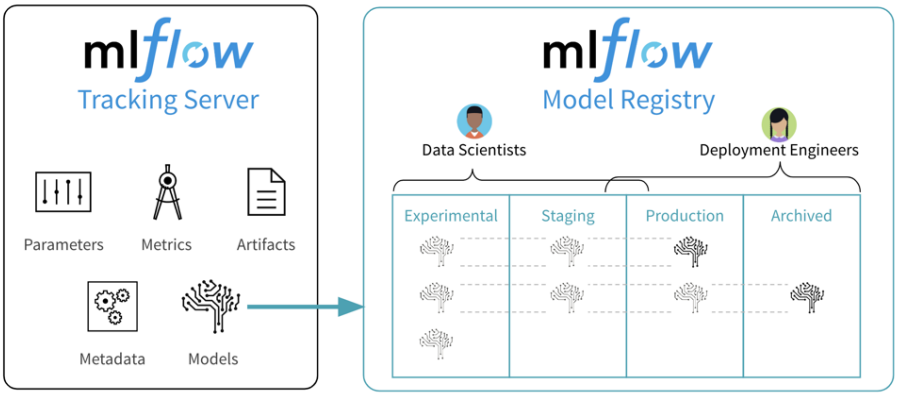

Figure 1: Components of MLflow [2]

MLflow is a free, open-source platform by Databricks which manages machine learning models from end-to-end. It is interoperable with a variety of machine learning libraries, cloud platforms, and programming languages. MLflow enables data scientists and machine learning engineers to train models, reuse them, and deploy them so that other scientists can utilize them as a "black box." It achieves this by packaging up models into reproducible steps. MLflow offers four major components (see figure 1):

- MLflow Tracking: This is an API that enables users to log parameters, user-defined metrics and output files when training a machine learning model. These results can be visualized using the MLflow UI.

- MLflow Projects: When tracking metrics and parameters using MLflow Tracking, a directory containing descriptor files is created. This file indicates dependencies of the model and how to execute the code. For instance, a file called 'conda.yaml' is created, which specifies the Python Conda environment used to train a model.

- MLflow Models: MLflow produces a directory containing arbitrary files and the actual model that was trained in the directory's root after a model has been trained. The files in the directory specify different flavors to view the model, such as "python function," "pytorch," "sklearn," etc.

- MLflow Registry: This component provides users with centralized model storage, a user interface, and APIs to manage the machine learning lifecycle of the model. It also gives information on which training experiment of the model produced particular results, and version control of models.

Kubernetes is an open-source container orchestration and control platform that automates the deployment, scaling and management of containerized applications. It is widely used in cloud computing for managing containerized application in production. We use MLflow to log parameters and metrics when training a deep learning model on Kubernetes clusters deployed on three different cloud platforms.

1.3 Why MLflow?

As mentioned earlier, another popular MLOps tool used is Kubeflow. Kubeflow is an MLOps tool that focuses on deploying models on Kubernetes clusters. It has the ability to run on any cloud platform that offers Kubernetes. However, it is deemed to be more complex than MLflow since along with the machine learning deployment workflow, it also handles contain orchestration. To use Kubeflow, one needs to have advanced technical knowledge to set up tracking metrics, for which we would have to use its metadata feature. But MLflow has this as a built-in feature. When it comes to deployment on Kubernetes, there are a lot of studies that show the deployment process using Kubeflow, but very few that use MLflow and compare the deployment process on different environments.

Based on our research, we would like to highlight the following advantages to using MLflow over Kubeflow:

- Ease of use: MLflow is generally easier to set up and use than Kubeflow. It is also more user-friendly, with a simpler interface and clear documentation.

- Versatility: MLflow can be used with a wide range of machine learning libraries and platforms, whereas Kubeflow is more specialized and geared towards Kubernetes.
- Integration: MLflow integrates with a large number of tools and platforms, making it easy to use in a variety of environments. Kubeflow, on the other hand, is tightly integrated with Kubernetes and may not work as well with other systems.
- Cost: MLflow is an open-source tool, whereas Kubeflow may require the use of paid resources such as Google Cloud Platform.

We wanted to experiment with an MLOps tool which could also be used to deploy models on Kubernetes, which provides many different deployment options and functionality, and doesn't involve a steep learning curve. MLflow may be a better choice for users who want a more general-purpose machine learning platform that is easy to use and integrates well with a variety of tools and systems. Kubeflow may be more suitable for users who specifically need a platform that is optimized for Kubernetes and are willing to invest the time and resources required to set it up and use it effectively. Therefore, we picked MLflow over Kubeflow for our experimentation.

2 Related Works

We conducted a literature survey of deploying ML models using various MLOps tools. A study conducted by Hewage et. al. compared the features offered by different MLOps tools, two of which were Kubeflow and MLflow [5]. MLflow seemed to offer more features than Kubeflow. The only feature that Kubeflow offered that MLflow didn't was Performance Monitoring. However, Kubeflow did not provide ways to perform data versioning and pipeline versioning, both of which MLflow offers.

Another study by Köhler compared the ability of three different MLOps tools that were designed for Kubernetes, to integrate with Kubernetes: Kubeflow, Pachyderm, and Polyaxon [6]. Their performance, vitality, usability and functionality were compared. The three technologies provide methods for orchestrating workflows and have the necessary features to manage the ML lifecycle in an unified environment. When it came to usability, the study concluded that a high degree of technical expertise is required even for a simple model deployment workflow, which would, on average, beyond the scope of a data scientist. Good knowledge of cloud network management is also required for setup; therefore usability was considered to be low. Additionally, the study found that there was a lack of up-to-date documentation for Kubeflow features, but Kubeflow and Pachyderm have considerable open-source community support, which is definitely a plus point. This study did not include MLflow since MLflow is not exclusive to Kubernetes, but MLflow mitigates the usability problem.

3 Experimental Setup

3.1 Aim

Our aim is to present a comparative study of deploying a deep learning model using MLflow on Kubernetes clusters in three popular cloud platforms: IBM Cloud, Google Cloud Platform, and Amazon Web Services. Deep learning models have gained immense traction in today's landscape. One of the most popular use cases of deep learning models is for the task of image classification. Hence, we deploy an image classification model with MLflow tracking and measure performance metrics during its training to compare the three cloud platforms. Details on the deep learning model used and the dataset it is trained on are elaborated in the following sections.

3.2 Dataset

We needed a publicly available dataset to train an image classification model. Furthermore, we needed to ensure that the images in the dataset are not too big (i.e., they have low, but sufficient resolution) so that training the models would not take too long since the nodes in the Kubernetes clusters we created for this experiment do not have GPU (since GPU enabled clusters cost significantly higher). A popular dataset that satisfies these prerequisites is the CIFAR-10 dataset, which is a collection of images that is used for object recognition. It contains 60000 32x32 color images with 10 mutually exclusive classes: airplane, automobile (but not truck or pickup truck), bird, cat, deer, dog, frog, horse, ship, and truck (but not pickup truck) [8]. Each class has 6000 images with a split of 5000:1000 training and testing batches that are randomly selected. This dataset is relevant to our scope due to its reasonable size and ease of scaling.

3.3 Model Used

To compare the performance of the deep learning model on Kubernetes clusters on different cloud platforms, we needed a model that has (i) high computational complexity, and (ii) high disk intensity. This allows us to compare model performance in terms of data load time on the different platforms, as well as to get an accurate picture of how long training the model would take in the three clouds. Therefore, we use the AlexNet model [7]. This model revolutionized the field of computer vision. It consists of eight layers, out of which the first five layers were convolutional layers, some of which are followed by max-pooling layers. The final three layers are fully connected layers. This model has sufficient depth for our experiment and is not too complex that training using CPU would take a significant amount of time. The model we use in our experiment was constructed using the PyTorch library [12], which MLflow also has support for.

3.4 Metrics

The metrics we measured were the following:

- Data Load Time: How much time it takes to load the dataset.
- Run Time: The amount of time it takes to run the entire job.
- Test Loss: Measure the model's performance during training.
- Training Time: The amount of time it takes to train the model.

3.5 Kubernetes Cluster Configuration

To make the comparative study a valid one, the configuration of the Kubernetes clusters we deploy in the three cloud platforms must be comparable with each other, i.e. they must have (i) the same number of nodes, (ii) created with similar VPC configurations with equal number of subnets, and (iii) be using similar operating systems. We also needed to keep in mind the cost of creating the cluster. Thus, we decided on the following configuration when creating the clusters:

- All three clusters have 3 nodes, with 2 vCPUs and having memory around 12 GB.
- We configured the cluster to use Linux operating system with Ubuntu 18 OS image.
- The clusters are created with a VPC with 1 subnet which is attached to a public gateway.

3.6 Architecture

Even though the experiment involves three cloud platforms, there will be many common elements in the process of training the model with MLflow tracking since the deployment is essentially on Kubernetes clusters. The general components involved in training the MLflow integrated model and viewing performance metrics using the MLflow UI are depicted in figure 2.

First, we need to integrate MLflow Tracking in our python code in in such a way that it stores the MLflow run data from a PostgreSQL database in the cloud platform. MLflow allows users to store model run metrics in various format such as PostgreSQL, SQL, among others, but PostgreSQL is widely used and is a standard. Next, we deploy the MLflow server using a Kubernetes service and deployment in the cluster. This is the UI where we will see the outputs of every model training job that is executed. The command to get the server running can be seen in the bottom of figure 2. The --backend-store-uri and –-artifacts-destination arguments lets the MLflow server know that the MLflow metrics data and model are stored in the PostgreSQL database and bucket storage respectively. Once the server is running, we will be able to visualize the

logged data for each training job that we run in the UI (Steps are detailed in Section 4).

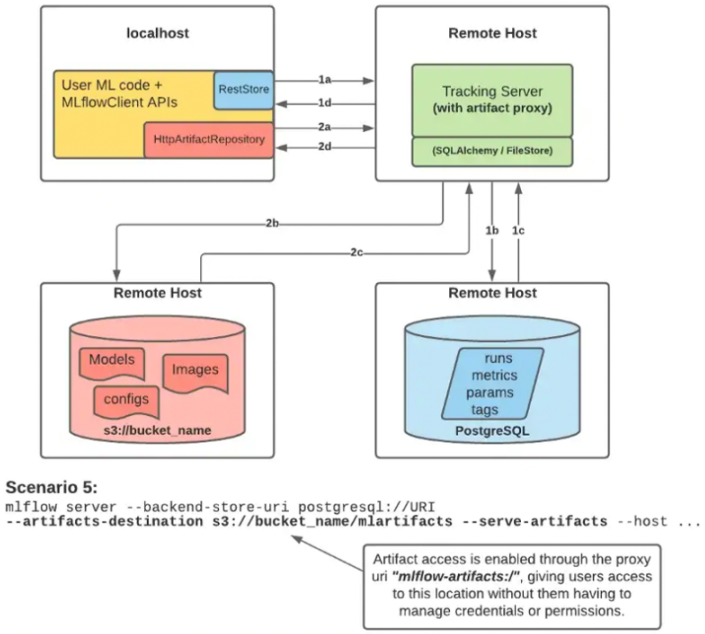

Figure 2: Deployment Architecture [11]

4 Deployment

In this section, we highlight the end to end steps and the code approaches for the deployment process.

4.1 MLflow Setup and Code Approach

The functionality to track experiments using MLFlow can be set up with a few simple steps. To set up the tracking server, follow these steps:

- Import the mlflow library
- Call the following function with the address of the tracking server as an argument. This is where the logs and tracked data will be stored. The server can be located locally or remotely.

  ```
  set_tracking_uri(...)
  ```

- Call the setup(...) function with the following additional parameters:
  - log experiment: set to True to enable logging of all parameters and metrics on the MLFlow server
  - experiment_name: specify the name of the experiment that you want to log. If not set, the default name is clf
  - log_plots (optional): set to True to log specific plots as png files
  - log_data (optional): set to True to log the training and test datasets used in the experiment"

```
with mlflow.start_run():
    runtimestart=time.time()
    main()
    runtimeend=time.time()
    mlflow.log_metric("runtime",str(runtimeend-runtimestart))
```

Figure 3: Code Approach

4.2 MLflow UI

After running your experiments, you can use the MLFlow UI to view the logs and analyze your results. The UI allows you to visualize, search, and compare experiments, as well as download models, data, and metadata for further analysis. To access the UI, you can either run the 'mlflow ui' command from the terminal or the '!mlflow ui' command from a notebook if the tracking server is a local folder. If you have not explicitly set a tracking server, the logs will be stored in the 'mlruns/' folder in your current directory. In this case, you can run the 'mlflow ui' command from your current directory and access the UI at 'http://localhost:5000'. If you are using a remote tracking server, you can access the UI by going to 'http://ip-address-of-tracking-server:5000'.

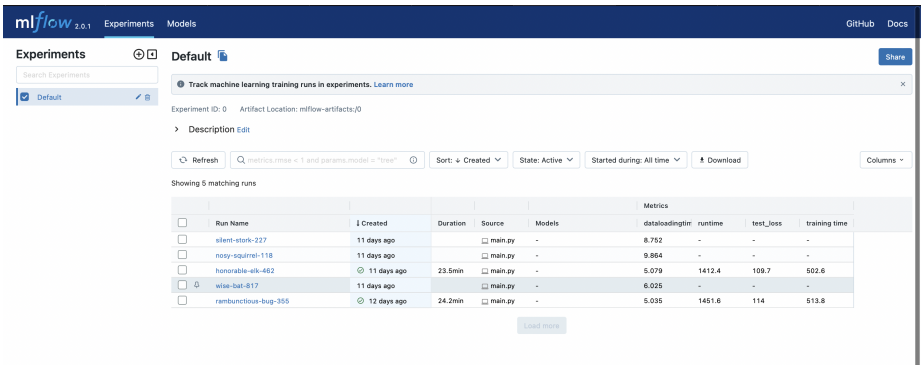

Figure 4: MLFlow UI

4.3 Docker Containerization

Before we begin the actual deployment onto the clusters, we would need to be able to create two of images, one would have the model, and the other would have the ml flow server. Once you have docker setup, these are the commands you would need to run. A docker account setup is necessary for this. The docker file used also has to be configured according to the cloud platform and the services that are being used.

```
#!/usr/bin/env bash
FROM python:3.8
RUN pip install mlflow
RUN apt update && \
        apt install -y gcc libpq-dev && \
        pip install psycopg2==2.8.5

EXPOSE 5000

CMD mlflow server --backend-store-uri postgresql://
ibm_cloud_e77054fd_0c9c_4c04_a3e1_5b787d1a29cc:4a67e4e3b5f5191b6cbe93
131bce19990bab322d28b71dbbcb2e4770253b3425@c8e97bd2-0444-4e53-80b0-3a
1a99b910c0.3c7f6c12a66c4324800651be37a77ceb.databases.appdomain.cloud
:30903/ibmclouddb?sslmode=require --host 0.0.0.0 --port 5000
```

Figure 5: Docker File Configuration

Once that is complete, now you are able to begin the setup of each of the cloud providers, and be able to create the necessary deployment files to be able to run the ml flow application and the model onto the separate clouds.

4.4 IBM Cloud

On all clouds, the setup approach remains similar. In this section, these steps are detailed and the things that had to be done specifically for IBM Cloud are also highlighted.

1. Create a cluster with a VPC with public gateways attached to the subnets in the VPC. The Kubernetes cluster follows the configuration details mention in the earlier section.

2. To set up the MLFlow server, you will need to create a cloud storage object in IBM Cloud and a Postgres database for storing MLFlow artifacts and performance metrics data.

3. Next, set up the MLFlow server with postgres linking and user access.

4. On IBM Cloud platform, this involves creating a user and obtaining the necessary keys and authentication. After creating a username, the service credentials are downloaded from the "Connect Using PostgreSQL Client" tab.

5. Finally, train the AlexNet model in the Kubernetes cluster.

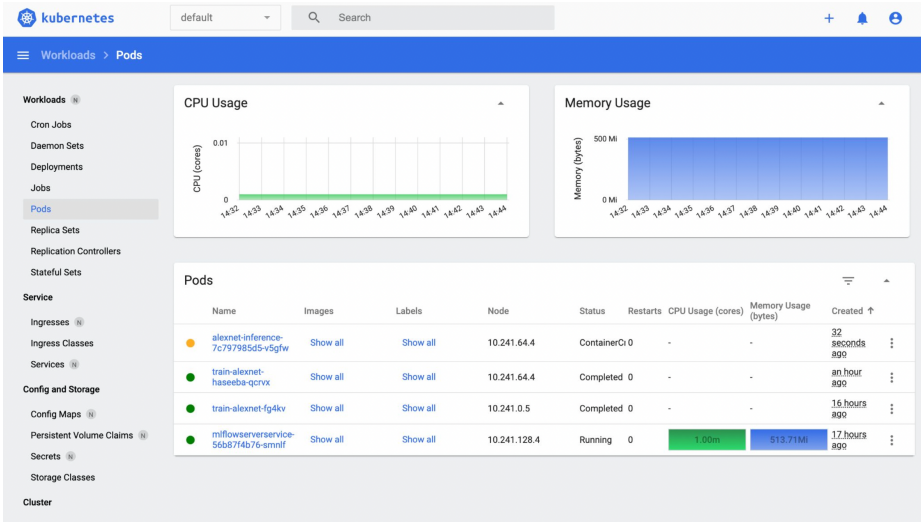

Figure 6: Kubernetes Cluster and Train Jobs on IBM Cloud

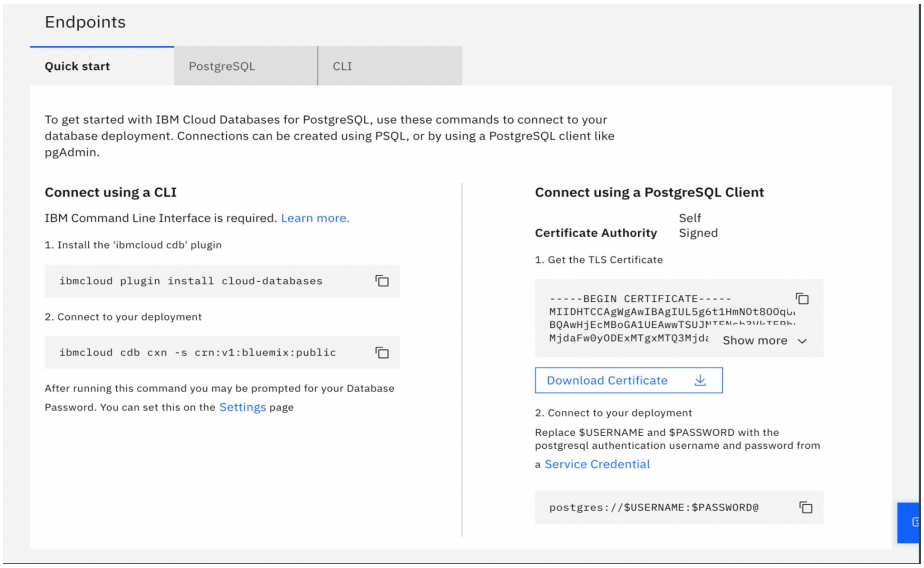

Figure 7: Creating a User and Obtaining PostgreSQL Credentials

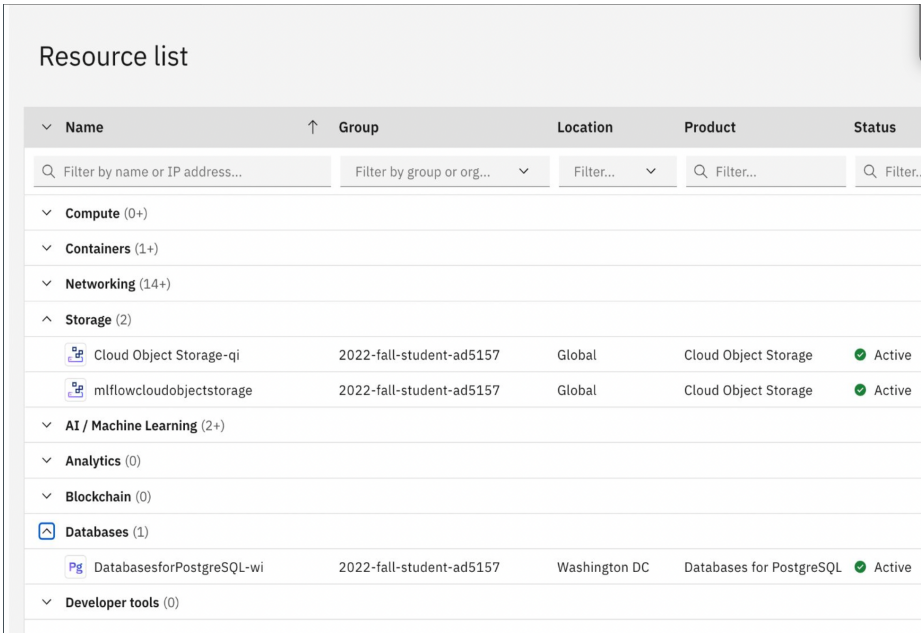

Figure 8: Resource List on IBM Cloud

4.5 Amazon Web Services

The process for deploying the MLflow Server UI and the train job in AWS was similar to the ones for IBM Cloud and Google Cloud Platform, however, we faced some challenges that were specific to AWS. Details of the deployment process are given below:

1. First, the Kubernetes cluster was created using Amazon Kubernetes Service [1]. The cluster configuration was the same as specified in Section 3.5. Cluster creation took less than 10 minutes. While making the cluster, we need to ensure that we create it with a VPC that is attached to a public gateway so that the MLflow server can be accessible to outside users.

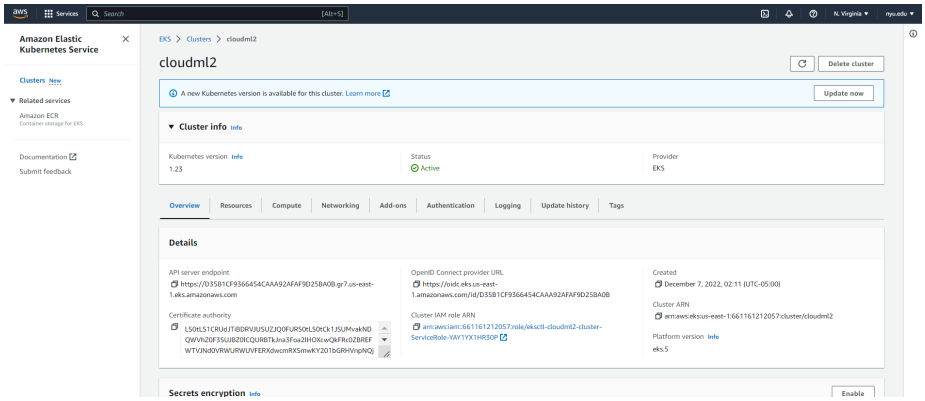

Figure 9: AWS - Kubernetes Cluster

2. After creating the cluster, we needed to create the necessary cloud resources for deploying the MLflow server, first of which was the PostgreSQL database. This was created using Amazon RDS, which is a fully managed database service that AWS offers. A new database instance can be set up in minutes and it is easy to set up, operate and scale. The database instance created and its configuration can be seen in figure 10.

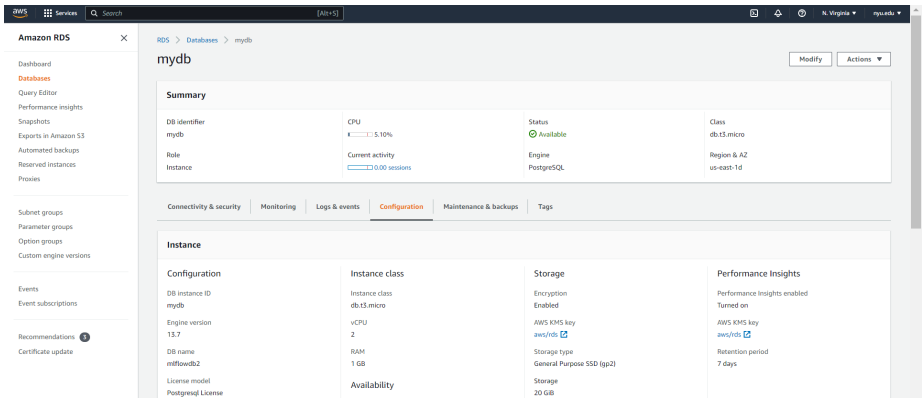

Figure 10: AWS - RDS Instance

3. Next, we need an Amazon Simple Storage Service (S3) bucket to store MLflow artifacts. S3 offers high data availability, scalability, high performance and security. Data can be stored, retrieved, and managed in the cloud with ease due to the robust and flexible Amazon S3 storage service.

4. Now that we have all the necessary resources, we can start the MLflow server with the Amazon RDS database as the backend store URI, and the S3 bucket as the artifacts destination in the Dockerfile for starting the MLflow server. We need to create .yaml files for the "service" and for "deployment" so that the UI is publicly accessible. Before doing so, we need to set the necessary IAM roles so that the services can interact with each other.

```
CMD ["mlflow", "server", "--backend-store-uri",
"postgresql://postgres:postgres@mydb.csht1fvg6xtn
.us-east-1.rds.amazonaws.com:5432",
"--default-artifact-root", "s3://cmlproj2bucket",
"--host",  "0.0.0.0" ,
"--port","5000"]
```

In AWS, getting the MLflow server service to connect with our PostgreSQL database. This was because, due to some reason, the PostgreSQL database was not accepting connections from any IP address, even though the same VPC was used to create the database and cluster. This was one of the main challenges we faced when deploying to AWS which made the process quite tricky. To mitigate this issue, we had to modify the routing rules of the VPC to accept connections from everyone by specifying the address 0.0.0.0. Then we can apply the yaml files for the Service (which used a LoadBalancer to provide hosting), and for the Deployment which ran the "mlflow server" command using the previously created service.

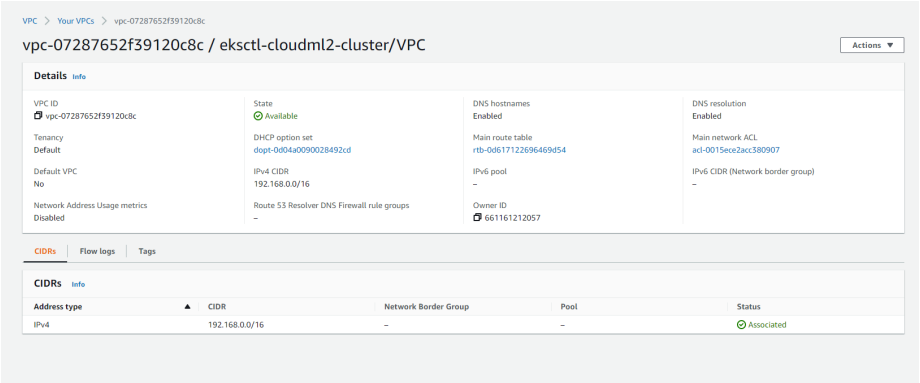

Figure 11: AWS - Adding VPC Routes

5. We can see the hosted MLflow UI. Each time we train the AlexNet model (or any model train job) following the same procedure as GCP and IBM Cloud, the UI will display the metrics logged of that particular train job and model in the UI.

4.6 Google Cloud Platform

1. Once you have the service account ready, the first thing you would need to do is to be able to generate a service key so that you can be able to access the service account locally from your terminal and be authenticated. You would also need to provide the relevant permissions of "Storage Object Admin" and the "Cloud SQL Editor" in the IAM process when you create the specific account.

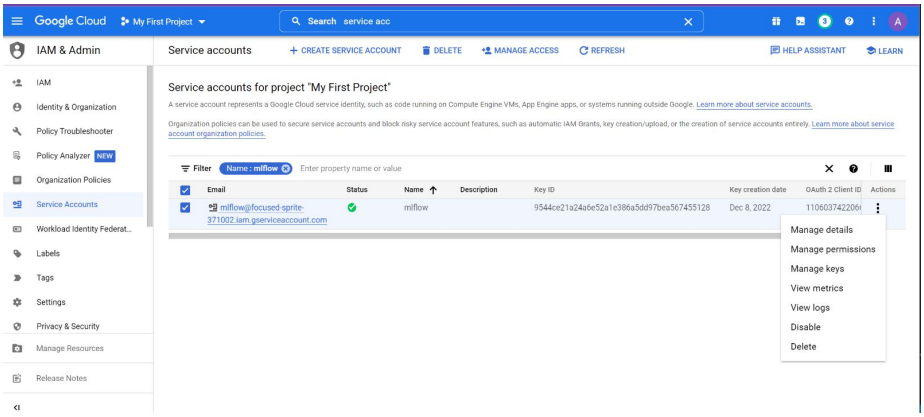

Figure 12: IAM on GCP

2. The next step is to create a bucket under the cloud storage option. A

bucket is essentially a way to store data in the cloud, it will be very important later when one is using ML flow to be able to access the bucket. You would need to create a standard bucket, and you can keep it multi region with no public access.

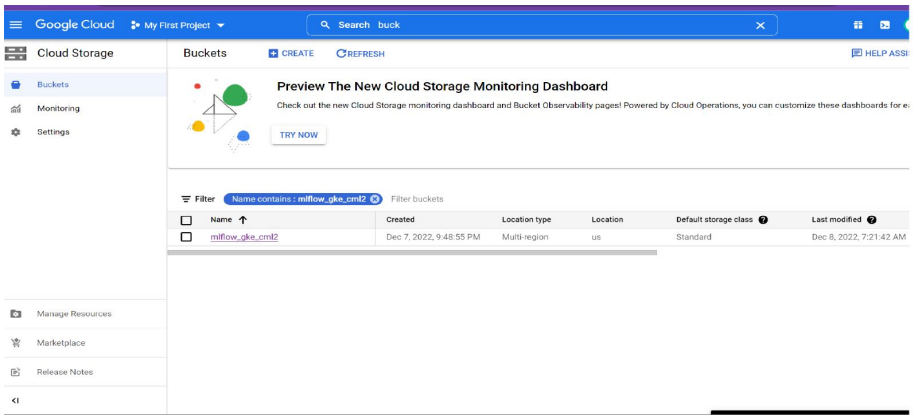

Figure 13: Cloud Storage Buckets on GCP

3. Now is time for the database through postgreSQL where you would need to create a postgreSQL database, in which one would need to set the name, and the username and password. Make sure to have access to all these three values because it will be useful at a later point.

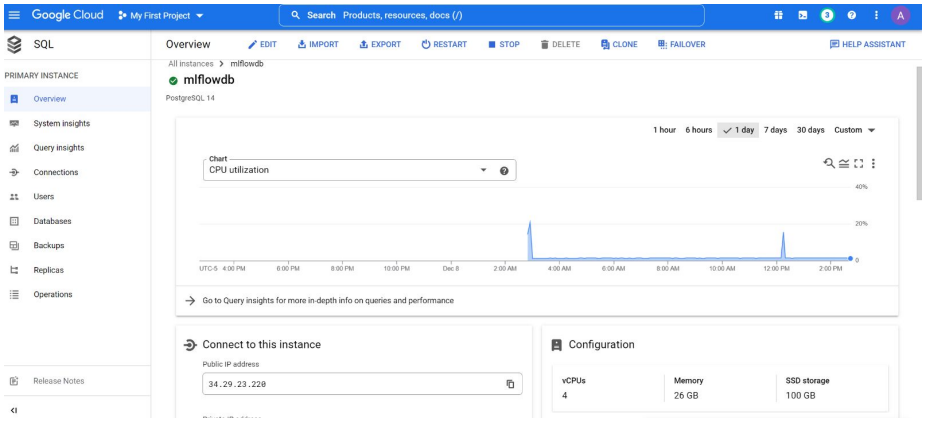

Figure 14: PostgreSQL on GCP

4. After that, you will create the actual Kubernetes cluster. Make sure to have the similar setup for all the clouds.

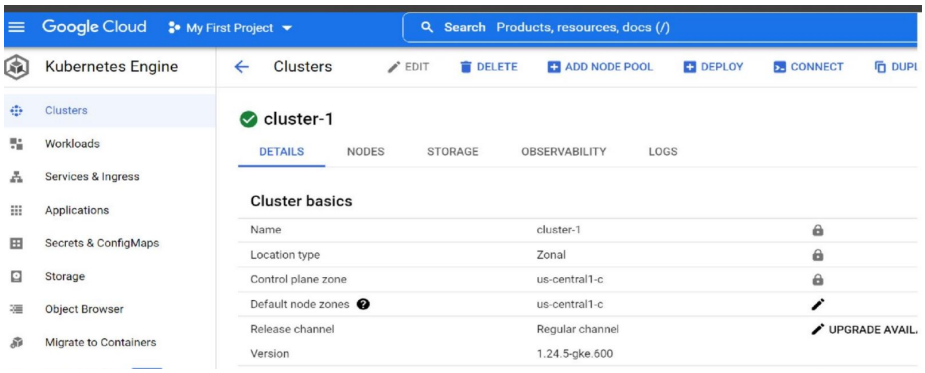

Figure 15: Kubernetes on GCP

5. Now once you have all of that setup, it is time to actually deploy and start using the already created YAML files using the kubectl command. You need to make a few minor changes. One is to make sure you change the mount paths to /var/lib and the other was to make sure you change the PostgreSQL database and add the name, username, password and bucket location.

Now that all the required resources have been set up, we can run the below commands for GCP and use kubectl to apply the .yaml files needed to set up the MLflow server. Once everything is deployed, you should be able to get the URL where the service is and be able to see the ML flow dashboard.[4]

```
gcloud auth activate-service-account
--key-file=file_from_iam.json

gcloud container clusters get-credentials cluster_name
kubectl create secret generic
gcsfs-creds --from-file=./keyfile.json
kubectl apply -f YAML file
```

5 Serving the Model

After each train job of the AlexNet model, the model trained gets saved by MLflow. This model was served through a flask application so users can run inference. The UI was fairly simple. We had a list of images (see figure 16) from which a user can provide an image index for which to run inference on. The Flask application loads the saved model, uses it to get the predicted classification of the image selected by the user, and returns the result to the UI.

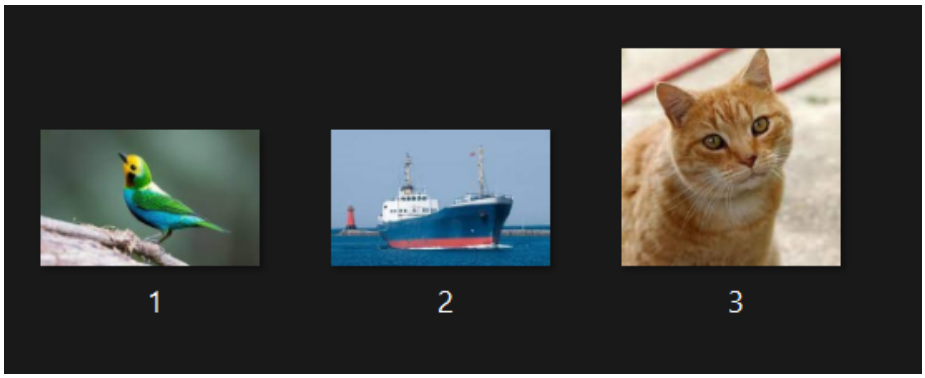

Figure 16: Image list for serving the model

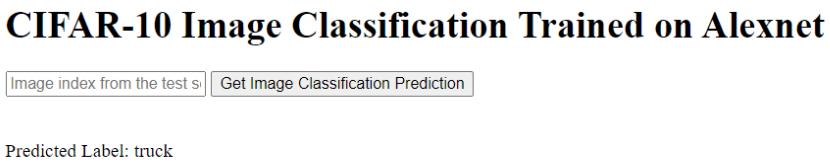

Figure 17: Flask UI

6 Results

In the following section, we highlight the performance metrics obtained from running the models on all the different cloud deployments.

6.1 Performance Metrics

Performance Metric	IBM Cloud	Google Cloud	AWS
Data Load Time	5.079	7.157	5.90
Run Time	1412.4	4181	1592
Test Loss	109.7	119	115.1
Training Time	502.6	1494.8	577.84

6.2 Results Analysis

The data load time for each cloud platform using a consistent set of data from the CIFAR-10 dataset and AlexNet to ensure that the comparison is accurate. Based on the table above, we can see that the data load time ranges from 5 to 7 seconds in each of the cloud platforms. The runtime is lowest for IBM Cloud

but is highest for GCP at 4181 seconds. Comparing the test loss, it seems to be consistent and this can be seen in Plot 20.

The training time again is similar for IBM Cloud and AWS, but higher for GCP. While researching what can cause the difference of the numerical results of these performance metrics we found that the difference in services provided by each cloud platform,infrastructure of the components and internal data processing tools may be an explanation for the variation in these times. The graphical results obtained from running the models on all the different cloud deployments are highlighted in the figures presented in this section.

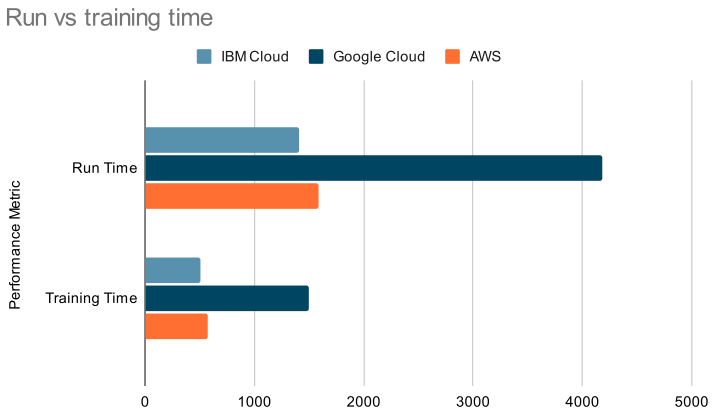

Figure 18: Run vs training time

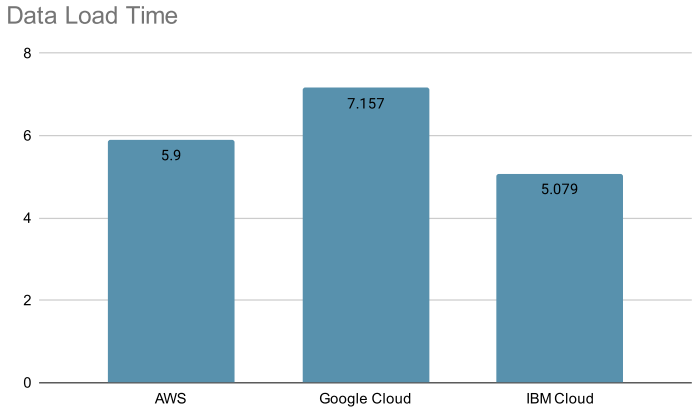

Figure 19: Data load times

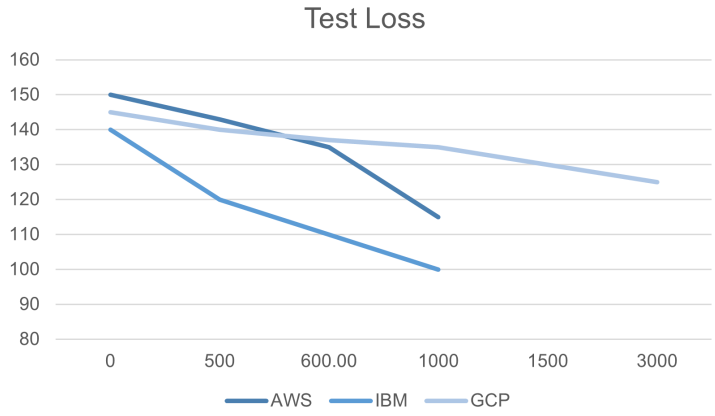

Figure 20: Test Loss

In this section, we highlight the findings of this analysis and evaluate the relative performance and ease of use of MLflow on each of these platforms. In the following discussion, we will also summarize the key findings and provide some insights into the factors that may influence the choice of cloud platform for MLflow deployment.

6.3 Ease of MLflow Deployment

Based on our study, MLflow is a user-friendly tool that is designed to make it easy for data scientists and machine learning engineers to track and manage their machine learning experiments.

It provides a simple API for logging and querying experiment data, as well as a web UI for visualizing and comparing results as showcased in this report. MLflow also integrates with a wide range of machine learning libraries and platforms, making it easy to use in a variety of environments. We determined that the it makes it very easy for us to be able to organize all the the different parameters and track with unique IDs. There is support for multiple frame works. Overall, MLflow is a convenient and intuitive tool that is well-suited for managing machine learning experiments at scale.

6.4 Cloud Deployment Comparison and Choice

The three cloud platforms used in this study (AWS, IBM Cloud, and GCP) are popular choices for deploying machine learning models, and all of them offer a wide range of tools and resources for building and deploying ML models.

Based on our experience, IBM Cloud had the least amount of documentation, but was most familiar for us to use. According to the results of our study, it was also well suited to the particular model we chose to deploy and the constraints of our project. In general, AWS and GCP tend to be popular choices

for machine learning workloads due to their extensive offerings of cloud-based machine learning services and infrastructure. IBM Cloud also has a strong focus on machine learning, with a range of services and tools specifically designed for building and deploying ML models.

It is difficult to make a general recommendation about which cloud platform is the best choice for deploying a model using MLflow, as the optimal choice will depend on a variety of factors such as the specific needs of the model, the resources and expertise available, and the cost constraints of the project. For future reference, it may be helpful to evaluate each platform based on factors such as the cost and availability of resources, the ease of use and integration with other tools, and the level of support and documentation provided before getting started.

6.5 Challenges

There are several challenges that we encountered when deploying a model using MLflow:

1. Lack of documentation for MLflow specifically for the IBM Cloud platform. There was no process/steps to follow and in our case, we used our knowledge of Kubernetes and IBM Cloud to eventually be able to do it.

2. Integrating the different services offered by cloud platforms was a challenging task, as each platform has its own unique set of tools and capabilities that may require different approaches to integration. The tools and capabilities of each platform was different and understanding them and the differences between the individual services themselves was very time consuming.

3. Another limitation we had was that we were restricted accessing few configurations while setting up since we had to keep in mind the cost estimates due to the tier level of most of the accounts. While we had access to IBM Cloud, we did not have access to AWS or GCP and had to buy the resources we were using. In general, deploying a model at scale can be resource-intensive, and it may be necessary to carefully manage resources such as compute, storage, and networking in order to ensure that the model performs optimally.

7 Conclusion

To conclude, this study highlights the deployment of deep learning models on Kubernetes clusters on various cloud platforms, and the use of MLflow to manage and track the performance of these models. In our analysis, IBM Cloud was found to have the fastest runtime among the three cloud platforms. Additionally, this study has allowed us to gain familiarity with Mlflow and learn about model deployment using MLflow in the cloud.

7.1 Future Work

There are several directions that future work could take based on the results of this study. For example, it would be interesting to extend the analysis to the following:

- Comparing the performance of different neural network architectures other than AlexNet potentially on tasks such as NLP can provide valuable insights into their performance with MLflow and the difference of performance among models.
- Additionally, documenting the process of deploying models using MLflow on various cloud platforms specifically like IBM Cloud which are relatively less explored can be useful for others who are interested in using the tool.
- Integrating with different AI platforms on the cloud, such as Azure ML Studio, can also be useful, as certain functionalities may make it easier to use. Platforms like Azure ML Studio also allow for no-code deployments and it would be interesting to see how this deployment process varies as compared to using Kubernetes.

Overall, these activities can help to expand our understanding of the capabilities and potential of using MLflow deployment for models and the tools available for managing and deploying them.

8 References

[1] Amazon. *Amazon Elastic Kubernetes Service (Amazon EKS)*. https://aws.amazon.com/pm/eks.

[2] Databricks. *Introducing the MLflow Model Registry*. https://www.databricks.com/blog/2019/10/17/introducing-the-mlflow-model-registry.html.

[3] DVC. *Data Version Control*. https://dvc.org/.

[4] Google. *Install kubectl and configure cluster access*. https://cloud.google.com/kubernetes-engine/docs/how-to/cluster-access-for-kubectl.

[5] Nipuni Hewage and Dulani Meedeniya. "Machine Learning Operations: A Survey on MLOps Tool Support". In: *arXiv preprint arXiv:2202.10169* (2022).

[6] Anders Köhler. *Evaluation of MLOps Tools for Kubernetes: A Rudimentary Comparison Between Open Source Kubeflow, Pachyderm and Polyaxon*. 2022.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[8] Hinton Krizhevsky Nair. "The CIFAR-10 dataset". In: ().

[9] Kubeflow. *Kubeflow*. https://www.kubeflow.org/.

[10] MLflow. *MLflow - A platform for the machine learning lifecycle*. https://www.mlflow.org/.

[11] MLflow. *MLflow Tracking*. https://www.mlflow.org/docs/latest/tracking.html.

[12] PyTorch. *PyTorch*. https://pytorch.org/.