iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 9 **Issue:** VII **Month of publication:** July 2021

DOI: https://doi.org/10.22214/ijraset.2021.36419

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue VII July 2021- Available at www.ijraset.com

Development of Kubeflow Components in DevOps Framework for Scalable Machine Learning Systems

Chandana EP[1], Nagaraj G Cholli[2]
[1]*IV Sem, MTech IT, Dept. of ISE, RV College of Engineering, Bangalore, India*
[2]*Associate Professor, Dept. of ISE, RV College of Engineering, Bangalore, India*

Abstract: ***Now-a-days, in the world of enterprise, machine learning workloads have become mainstream. However, there is an abundance of choices that can be made around multi-cloud infrastructure and machine learning toolkits, making it complex to balance their costs and performance. Microservices architecture has been the preferred architecture style for a few years now and there's been rapid growth in its adoption, never failing to provide exceptionally testable & maintainable services. To have a lot more simplified services management, deployment and to orchestrate tools, Kubernetes is recommended. Kubeflow, a known and widely adopted open source container management platform that manages machine learning stack on Kubernetes. This paper discusses the development and validation of Kubeflow components such as PyTorch, TensorFlow, & Notebook Servers. It includes PodDefault functionalities for notebooks and container builder API to build docker images using Kaniko. Using Helm, Kubeflow upgrade operation is performed to enhance the configured resources whenever required for the distributed training jobs & workloads. Hence, providing data scientists a scalable platform to run machine learning workloads without having to worry about resources, costs, time, and portability.***

Keywords: Helm, Kubeflow, Kubernetes, Microservices, Distributed training.

I. INTRODUCTION

Kubeflow is a platform for developing and deploying a machine learning (ML) system. Data scientists use this platform to build and experiment with ML pipelines. Operational teams and ML engineers also use Kubeflow wanting to deploy machine learning systems to different environments for development, testing and serving at the production level. To detail the ML tools needed for the workflow, Kubeflow configuration interfaces can be used. Later, for both experimentation and production use, various clouds, on-premises, and local platforms are available to deploy workflow. There are several stages in machine learning workflow. An iterative process is involved to develop a machine learning system.

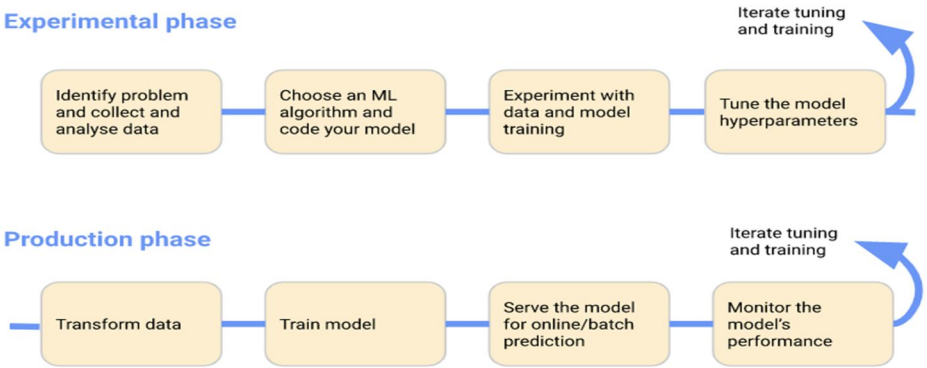


Fig. 1 Stages in ML Workflow [1]

Fig. 1 represents several stages of ML workflow. Its output must be evaluated and required changes to the model and its parameters is to be done whenever necessary to ensure the model keeps generating the required results. To mainly focus on the work of developing, training, testing, and deploying models in such a way that primary work is made possible by looking after issues like, faster and more consistent deployment, better control over ports and component access for tighter security, protection against over-provisioning resources, saving costs, and protection against tasks not being deallocated once complete, saving costs. The main objectives of this paper are to deploy and scale distributed machine learning models on Kubeflow, validate the training of distributed models, config upgrade Kubeflow and enable PodDefault functionality to notebook servers & validate container builder API to Jupyter notebooks.

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue VII July 2021- Available at www.ijraset.com

II. RELATED WORK

The Kubebench methodology consists of 5 major steps namely; Configuration step aids to frame a configuration of benchmark job, run Kubeflow job step, Compile Outputs step described the logs & outputs from the jobs achieved thus producing a result report, report results step and the last step is to clean up. Thus, enterprises can hold quantitative data with the help of Kubebench and to make upskilled judgement about models. [2]

With Teradata ViewPoint relating to the machine learning engine over HTTPS or HTTP, viewpoint facilitates Unified Data Architecture. All the containers of ML-engine interact with Heapster thus fetching statistics with the help of cAdvisor. The results show that the implementation of Heapster client and time- series based data being channelized to various subsystems will not load the system, for which the solution is built. [3]

Modeling techniques-based machine learning methods: Support Vector Machine, Artificial Neural Networks and Linear Regression is considered. These methods categorize the application's over memory, I/O resources, and CPU, running in the containers. Experimental results show that the SVM model's and ANN model's prediction accuracy is considerably better than Linear Regression based approaches. [4]

The methodology includes Hierarchical Architecture. It controls microservice's elasticity-based applications. Distributed & subordinated components' run-time adoption can be viewed at a higher level as per centralized application. Experiment results show that the default Kubernetes Auto-scaler is worse when compared to hierarchical control's benefits. [5]

III. THEORY AND METHODOLOGY

A. Kubeflow Architecture

Fig. 2 represents the system architecture of Kubeflow components. A set of services, tools and frameworks will be deployed together into one Kubernetes cluster, which enables end to end machine learning workflows. These Kubeflow services assist various workflows and are independently developed. It makes use of Istio Ingress providing a way to connect, secure & monitor services, gives a layer for supporting quotas, access controls, tracing the traffic including egress in the Kubeflow deployment. The Kubeflow components will get identity-based authentication, making it stronger deployment on the platform. Hence, this architecture acts as an underlying infrastructure that helps to deploy various services and components.

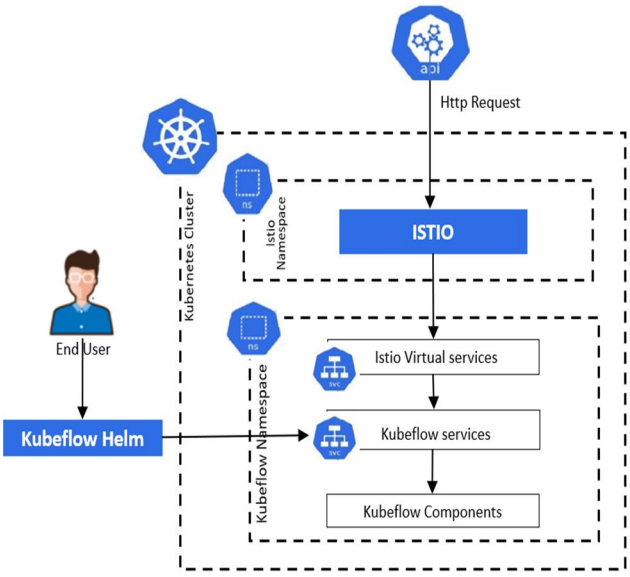


Fig. 2 Kubeflow Architecture

B. Methodology Adopted

Fig. 3 represents the methodology involved in development and validation of Kubeflow Components in DevOps framework.

Fig. 3 Methodology of Development and validation of Kubeflow Components in DevOps Framework

1) *Create Dockerfile:* Dockerfile includes all the commands to assemble an image as per user requirements. This Dockerfile can be run by using the "Docker build" command, which builds docker images with a context specifying the file PATH. It executes many command line instructions in a sequence. Here, Dockerfile for Pytorch, Tensorflow and Jupyter Notebook is created and built.
2) *Image Repository:* Image repository is a place or storage where a collection of container images is stored. It contains various versions of the same service or application. Here, the images built by the respective Dockerfile are kept.
3) *Create Pods:* Pods is created for Pytorch, Tensorflow and Jupyter Notebook with its related services/volumes to perform the training or to requirements needed by its respective functionalities. Pods created generally need to be in running state at least.
4) *Radish Framework:* For verification of the above specified components' functionalities, radish framework is used to check whether the job/training is happening successfully or not. This step is implemented in a file called "stepfile.py", where the steps required to verify are written accordingly as needed.
5) *Kubeflow pipeline:* Kubeflow pipeline helps to continuously integrate the code and detects the changes that occurred after a commit has been made in the source code repository. This pipeline is configured according to the user requirement for a component to run.
6) *Cucumber Report:* After the steps involved in building Kubeflow components and verifying it, the Kubeflow Pipeline produces a cucumber report mentioning the number of scenarios, test and steps that has successfully passed or failed, giving user an overall view of the behavior of the system considering the changes in the code

IV. IMPLEMENTATION

Fig. 4 represents the High level Sequence Diagram of Kubeflow Components Workflow. It shows how the objects interact and helps to identify how a particular use case performs when defined a particular set of events. It focuses on explaining how the Kubeflow development methods are involved, to achieve successful training or execution of jobs for the following PyTorch, TensorFlow and Notebook related functionalities.

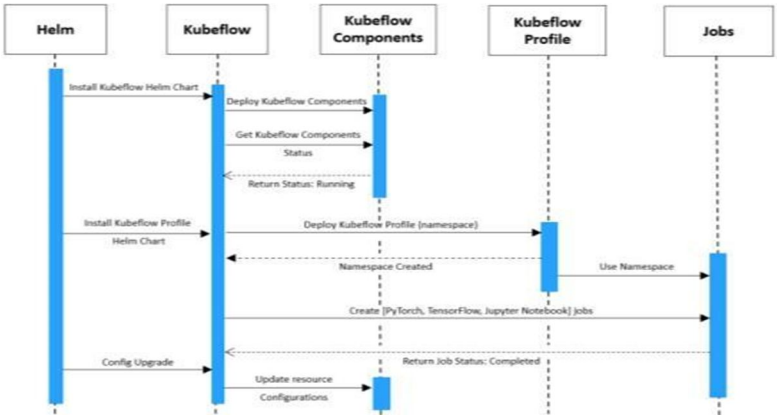


Fig. 4 High Level Sequence Diagram of Kubeflow Componentstages in ML Workflow

1) To create ***PyTorchJob***, PyTorch operator and Pytorch CRD must be created. Custom Resource is an object that extends Kubernetes API or allows users to create their own API into clusters whereas Custom Resource Definition (CRD) is a file used to define own object kind, handled by the API server throughout the entire lifecycle. Once the PyTorch operator is deployed by Kubeflow, PyTorch Jobs can be created many times. PyTorch focuses on providing better optimization, transparent model behavior, and better compatibility with NumPy and other python libraries. Once the PyTorch YAML (Yet Another Markup Language) file is created in the user namespace, the PyTorch pods creates the container and goes to running state. Fig.5 shows that PyTorchJob has worker nodes and master nodes to load balance the training and the moment training completes PyTorch pod goes to completed state.

```
[root@vm-10-75-47-8 chandana]# kubectl get pods -n chandana
NAME                                READY   STATUS      RESTARTS   AGE
pytorch-dist-mnist-gloo-master-0    0/1     Completed   0          4m29s
pytorch-dist-mnist-gloo-worker-0    0/1     Completed   0          4m28s
```

Fig. 5 PyTorchJob Pods Status

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue VII July 2021- Available at www.ijraset.com

2) *TensorFlowJob* (TFJob) YAML file is created in user namespace, then PyTorch pods creates the container of the image specified and goes to running state. The training happens by worker pods and replica sets which are created by pod template in specified numbers. Training happens using a simple SoftMax model with a hidden layer. The parameters (weights and biases) are located on one parameter server (ps), while the ops are executed on two worker nodes by default. The TF sessions also run on the worker node. The parameter updated from one worker is visible to all other workers. Likewise, workers can perform forward computation and gradient calculation in parallel, which should lead to increased training speed.

3) *Container Builder* is an API which builds docker images using Kaniko. Kaniko does not depend on docker daemon, executing each command within a docker file fully in a user space. Kaniko helps to unpack the base image with the right permissions. Kaniko enables container images to be built in environments that cannot securely run a docker daemon. This container builder requires a "ReadWriteMany" volume attached to the notebook for execution. The following values need to be provided for Container Builder:

a) A TensorFlow image as base image with non-root privileges.
b) PVC (Persistent Volume Claim) name needed to be attached to the notebook.
c) PVC path to mount the volume.
d) Secret name if exists and Service account name for container builder.

A fairing builder pod comes up when the container builder successfully runs the docker base image as shown in Fig 6.

Fig. 6 Test Docker build using Container Builder

4) For *PodDefault* functionality, a PVC must be created to mount the notebook volume in its path. PVC can be of glusterfs storage class with 'ReadWriteMany' access. Once the PVC is in a bounded state, the notebook must be created and configured to the above PVC. Both PVC and notebook must be created in the same namespace. Each notebook server belongs to a single namespace only. It must contain the memory requests and CPU requests that can be configured according to the user requirements. The notebook server can also be created with pykernel or rkernel machine learning kernel methods which are containerized. A PodDefault is created which describes the additional runtime requirements like environment variables, volumes and volume mounts that need to be injected into a Notebook Pod at creation time. PodDefaults uses label selectors to specify the Pods to which a given PodDefault applies. PodDefaults are namespace scope, i.e., they can be applied/viewed in the namespace. Fig. 7 shows the notebook server's environment list where the PodDefault enabled secret is provided.

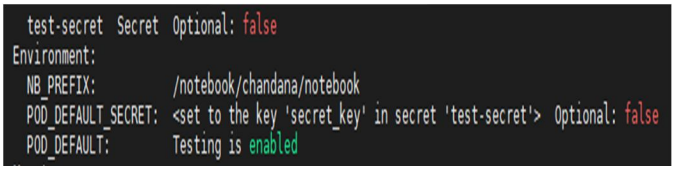


Fig. 7 PodDefault Configured Notebook

5) For *Config Update*, Kubeflow Helm chart needs to be installed with the required Kubeflow version. The Helm upgrade command overwrites the previous Kubeflow chart version and other resources passed with it as parameters, thus bringing up post upgrade installer pods in Kubeflow installer namespace with the below configurations.

```
componentResources:
  limits:
    cpu: "500m"
    memory: "1Gi"
  requests:
    cpu: "50m"
    memory: "128Mi
```

Kubeflow Helm chart argument can be either: a chart reference, a path to a chart directory, a packaged chart, or a fully qualified URL. The latest version of Kubeflow chart is specified with flag '-- version'. This can be achieved by using the command: *helm upgrade kubeflowinstaller kubeflow/ -f values.yaml --set componentsResource - Configuration.enabled=false --wait --timeout=1200s*

V. DATASET

MNIST (Modified National Institute of Standards and Technology) is a database. This dataset is used to train distributed training models in Kubeflow, just to test its framework, to ensure that they work. It contains handwritten digits from 0-9, & divided into training (60,000examples) and test set (10,000examples). These handwritten digits are normalized and centered in a gray scale image of 0-255, where each image is of the same size. Also, every image consists of 784 pixels that constitute the features of the digit.

VI. RESULT AND DISCUSSION

A. Detailed Analysis

The objective of this paper is to deploy and scale distributed machine learning models on Kubeflow by using Kubeflow Components such as PyTorch, TensorFlow and Notebook Servers, making production uncomplicated and simpler. Upgrading Kubeflow by configuring Kubeflow Component resources was successfully achieved. All the objectives were validated on upgraded Kubeflow too. Fig. 8 represents PyTorchJob Training & Accuracy. Accuracy metric is used in PyTorch to evaluate the distributed training of MNIST dataset using both worker and master pods to balance the load and resulting in accuracy of 96.62%.

Fig. 8 PyTorchJob Training Accuracy

Fig. 9 represents TFJob Validation Cross Entropy. "Cross Entropy" metric examines the predictions of models with the true probability distribution. It sends data efficiently to the other end/destination without or less loss of data. TFjob uses a cross entropy metric to evaluate the training consisting of a SoftMax model with a hidden layer.

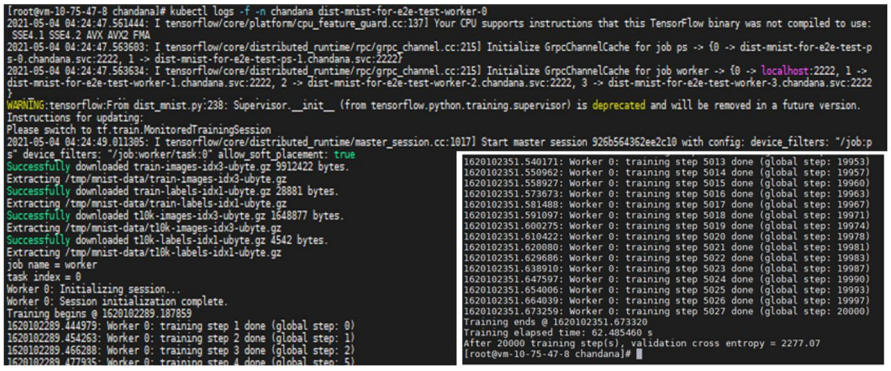


Fig. 9 TFJob Validation Cross Entropy

B. Performance Analysis

On average, the performance of Kubeflow depends on the resources and volumes available for creation of jobs. Particularly for Jupyter Notebook, the number of CPUs request must be configured according to the job executed by it, as CPUs requests and limits are associated with the container. Fig. 10 represents the Kubeflow components data dimensioning, which shows the consumption of CPU requests and memory used by each component.

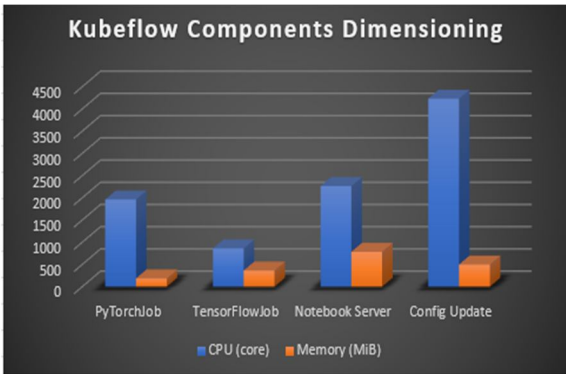


Fig. 10 Kubeflow Components Data Dimensioning

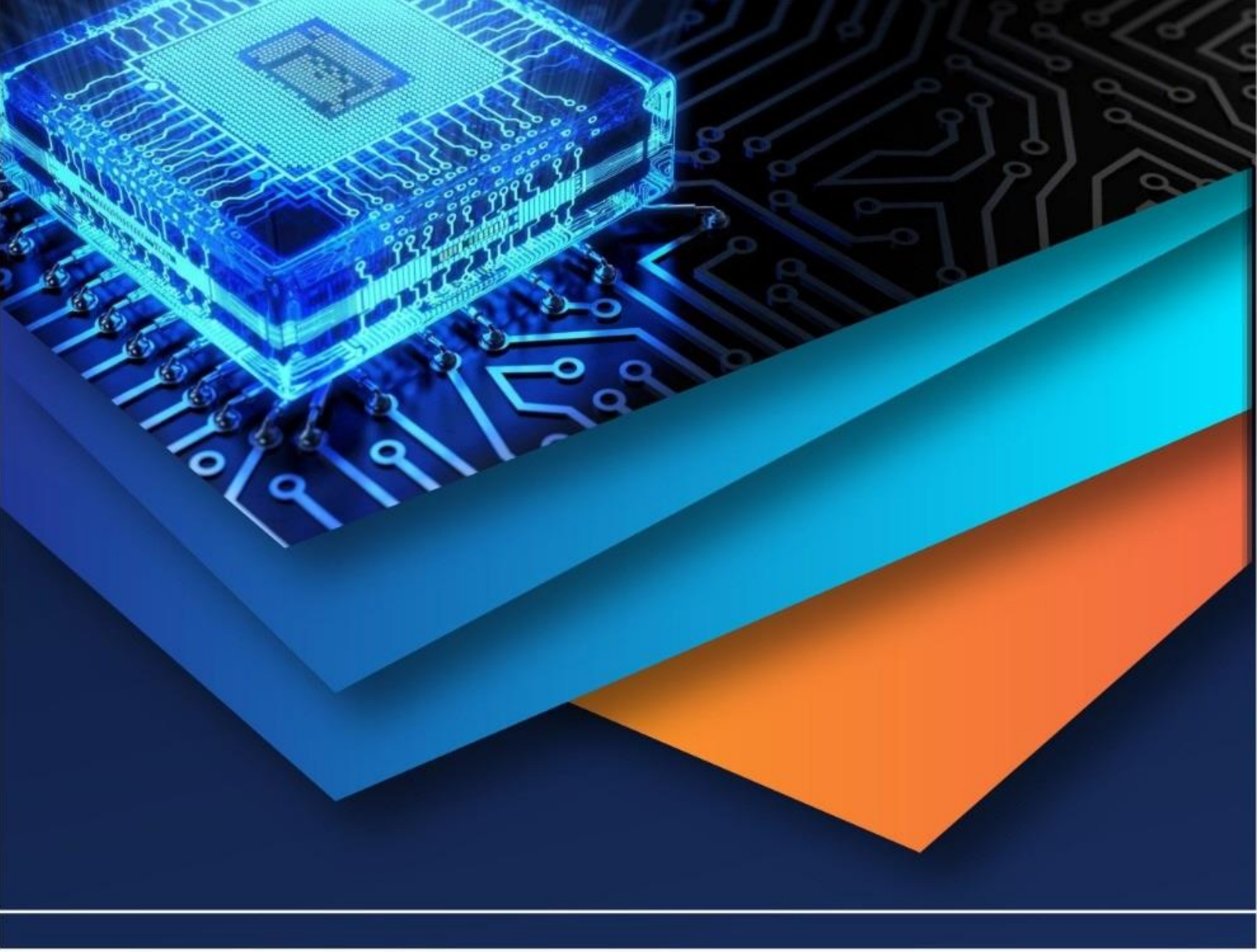
VII. CONCLUSIONS AND FUTURE WORK

A reference workload given to Kubeflow performs in a consistent and compatible way to assist machine learning life cycle by describing, running and collection of metrics. In a way, Kubeflow components assist to perform tracking and managing metadata of machine learning workflows and distributed training of models. Seamless integration between ML code, Jupyter notebooks, Kubernetes and ML libraries assists to easily containerize ML models requiring less amount of time than traditional methods of deploying ML models. The containerization of ML models is taken care for TensorFlow, PyTorch & Jupyter Notebook components and Kubeflow upgrade, thus helping to share the workload through pods and consuming very less volumes for storage and time to complete the distributed training, Fairing and upgrade tasks respectively.

As a part of future work, KeyCloak gatekeeper can be configured to Kubeflow applications to add better authentication and authorization, which enables security to front-end dashboard and backend services, networking functionalities can be enhanced using Istio CNI plugin, where users need not have to enable the elevated Kubernetes RBAC permissions.

REFERENCES

[1] The Kubeflow Authors, Documentation Distributed under CC by 4.0, 2021, https://www.kubeflow.org/docs/started/kubeflow-overview/
[2] X. Huang, A. K. Saha, D. Dutta and C. Gao, "Kubebench: A Benchmarking Platform for ML Workloads," 2018 First International Conference on Artificial Intelligence for Industries (AI4I), 2018, pp. 73-76.
[3] N. Parekh, S. Kurunji and A. Beck, "Monitoring Resources of Machine Learning Engine In Microservices Architecture," 2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2018, pp. 486-492.
[4] R. Muddinagiri, S. Ambavane and S. Bayas, "Self-Hosted Kubernetes: Deploying Docker Containers Locally With Minikube," 2019 International Conference on Innovative Trends and Advances in Engineering and Technology (ICITAET), 2019, pp. 239-243.
[5] A. Pereira Ferreira and R. Sinnott, "A Performance Evaluation of Containers Running on Managed Kubernetes Services," 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 2019, pp. 199-208.
[6] K. Ye, Y. Kou, C. Lu, Y. Wang and C. Xu, "Modeling Application Performance in Docker Containers Using Machine Learning Techniques," 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), 2018, pp. 1-6.
[7] L. P. Dewi, A. Noertjahyana, H. N. Palit and K. Yedutun, "Server Scalability Using Kubernetes," 2019 4th Technology Innovation Management and Engineering Science International Conference (TIMES-iCON), 2019, pp. 1-4.
[8] F. Rossi, V. Cardellini and F. L. Presti, "Hierarchical Scaling of Microservices in Kubernetes," 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), 2020, pp. 28-37.
[9] K. Ye and Y. Ji, "Performance Tuning and Modeling for Big Data Applications in Docker Containers," 2017 International Conference on Networking, Architecture, and Storage (NAS), 2017, pp. 1-6
[10] L. Cai, Y. Qi, W. Wei and J. Li, "Improving Resource Usages of Containers Through Auto-Tuning Container Resource Parameters," in IEEE Access, vol. 7, 2019, pp. 108530-108541.
[11] M. F. Bestari, A. I. Kistijantoro and A. B. Sasmita, "Dynamic Resource Scheduler for Distributed Deep Learning Training in Kubernetes," 2020 7th International Conference on Advance Informatics: Concepts, Theory and Applications (ICAICTA), 2020, pp. 1-6.
[12] M. Moravcik and M. Kontsek, "Overview of Docker container orchestration tools," 2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA), 2020, pp. 475-480.
[13] Y. Zhou, Y. Yu and B. Ding, "Towards MLOps: A Case Study of ML Pipeline Platform," 2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE), 2020, pp. 494-500.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY