Homework 5: ML and Kubernetes

Hrithik Dhoka
MS CS
New York University
hhd2023@nyu.edu

I. Introduction

The ever-growing field of machine learning (ML) relies heavily on efficient training and deployment of models. For computationally intensive tasks, distributed training approaches utilizing multiple computing resources are becoming increasingly necessary. However, managing these distributed environments can be challenging, requiring solutions that offer scalability, resource optimization, and ease of deployment.

This paper explores the use of Kubernetes, a popular container orchestration platform, for training, deploying, and performing inference on the MNIST handwritten digit classification dataset. The MNIST dataset, a well-known benchmark for image recognition, serves as a perfect example to showcase the capabilities of Kubernetes in a controlled environment.

We present a containerized workflow encompassing the entire machine learning lifecycle within a Kubernetes cluster. This includes containerizing the training script for distributed training on worker nodes, deploying the trained model as a service for real-time predictions, and utilizing Kubernetes features for efficient resource allocation and management. The code can be found here

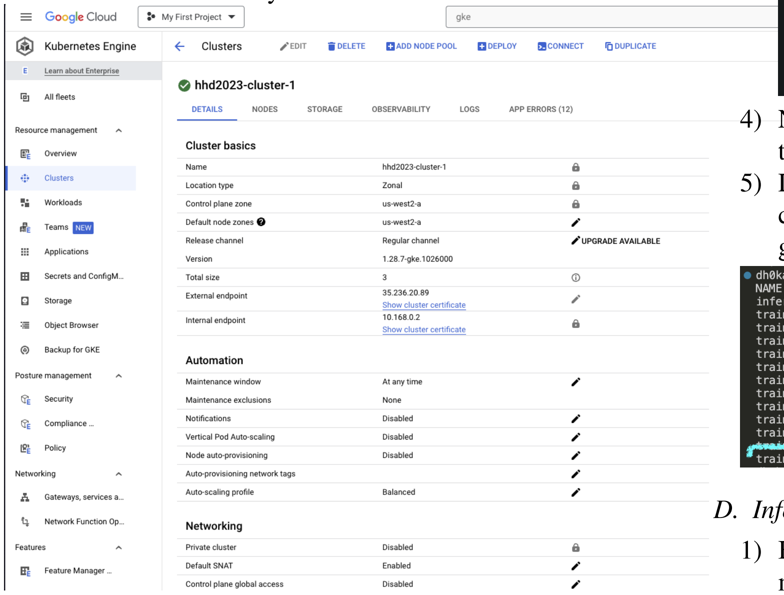
II. Methodology

A. *Creation of cluster on GKE*

We will then need to create a kubernetes cluster on Google Kubernetes Engine.

The kubectl will be configured in the cloud shell by the following command. gcloud container clusters get-credentials <kubernetes-cluster-name> −−zone <zone> −−project <project-name>

Now we can successfully use kubectl commands.



hhd2023-cluster-1

Cluster basics

Name	hhd2023-cluster-1	
Location type	Zonal	
Control plane zone	us-west2-a	
Default node zones	us-west2-a	
Release channel	Regular channel	UPGRADE AVAILABLE
Version	1.28.7-gke.1026000	
Total size	3	
External endpoint	35.236.20.89 Show cluster certificate	
Internal endpoint	10.168.0.2 Show cluster certificate	

Automation

Maintenance window	At any time
Maintenance exclusions	None
Notifications	Disabled
Vertical Pod Auto-scaling	Disabled
Node auto-provisioning	Disabled
Auto-provisioning network tags	
Auto-scaling profile	Balanced

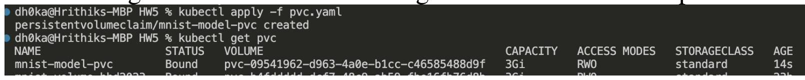
Networking

Private cluster	Disabled
Default SNAT	Enabled
Control plane global access	Disabled

B. *Creating and attaching PVC resource to the cluster*

1) To attach a PVC, we will need pvc.yaml which all specify all the configurations of our persistent storage.
2) Create a pvc.yaml in the cloud cluster by vim pvc.yaml.
3) In our cloud shell, we will use command kubectl apply -f pvc.yaml to apply these configurations to our cluster.
4) We can check using the command,kubectl get pvc or kubectl get pvc mnist-model-pvc whether our pvc has been successfully bounded with our cluster or not.

Thus, our PVC is ready now and successfully bound, we can go ahead with training and inference steps.

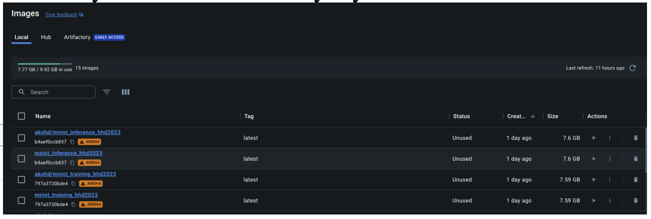


```
dh0ka@Hrithiks-MBP HW5 % kubectl apply -f pvc.yaml
persistentvolumeclaim/mnist-model-pvc created
dh0ka@Hrithiks-MBP HW5 % kubectl get pvc
NAME                STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
mnist-model-pvc     Bound    pvc-09541962-d963-4a0e-b1cc-c46585488d9f   3Gi        RWO            standard       14s
```

C. *Training program in kubernetes*

For training we will have to create a dockerfile, push the docker image to docker registry, and then create a training.yaml and apply that configuration to the cluster.

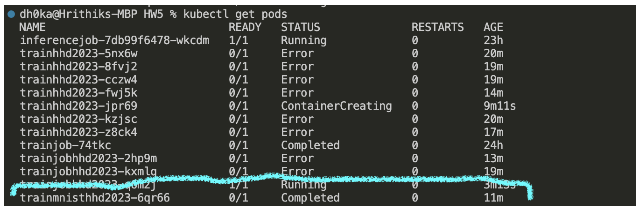
1) Build the docker image by docker build -t mnist_training_hhd2023 . –platform=linux/amd64. (This will name my image as mnist_training_hhd2023)
2) Then tag the image by docker tag mnist_training_hhd2023 akohd/mnist_training_hhd2023
3) Then push the train image to the dockerhub registry by using the command docker push akohd/mnist_training_hhd2023. Thus your train image is ready to be used by your kubernetes cluster.



7.77 GB / 9.92 GB in use 15 images

Name	Tag	Status	Creat...	Size	Actions
akohd/mnist_inference_hhd2023 b4aef0ccb837 AMD64	latest	Unused	1 day ago	7.6 GB	
mnist_inference_hhd2023 b4aef0ccb837 AMD64	latest	Unused	1 day ago	7.6 GB	
akohd/mnist_training_hhd2023 797a3720bde4 AMD64	latest	Unused	1 day ago	7.59 GB	
mnist_training_hhd2023 797a3720bde4 AMD64	latest	Unused	1 day ago	7.59 GB	

4) Now we will create train.yaml in the cloud shell by vim train.yaml.
5) Deploy the training job: kubectl apply -f train.yaml.To check whether training is successful we can see kubectl get jobs and kubectl get pods

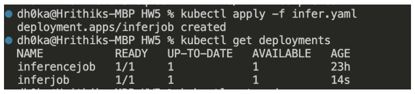


```
dh0ka@Hrithiks-MBP HW5 % kubectl get pods
NAME                          READY   STATUS              RESTARTS   AGE
inferencejob-7db99f6478-wkcdm   1/1     Running             0          23h
trainhhd2023-5nx6w            0/1     Error               0          20m
trainhhd2023-8fvj2            0/1     Error               0          19m
trainhhd2023-cczw4            0/1     Error               0          19m
trainhhd2023-fwj5k            0/1     Error               0          14m
trainhhd2023-jpr69            0/1     ContainerCreating   0          9m11s
trainhhd2023-kzjsc            0/1     Error               0          20m
trainhhd2023-z8ck4            0/1     Error               0          17m
trainjob-74tkc                0/1     Completed           0          24h
trainjobhhd2023-2hp9m         0/1     Error               0          13m
trainjobhhd2023-kxmlq         0/1     Error               0          19m
trainjobhhd2023-qom2j         1/1     Running             0          3m13s
trainmnisthhd2023-6qr66       0/1     Completed           0          11m
```

D. *Inferencing (trained program) in kubernetes*

1) Build the docker image by docker build -t mnist_inference_hhd2023 . –platform=linux/amd64.

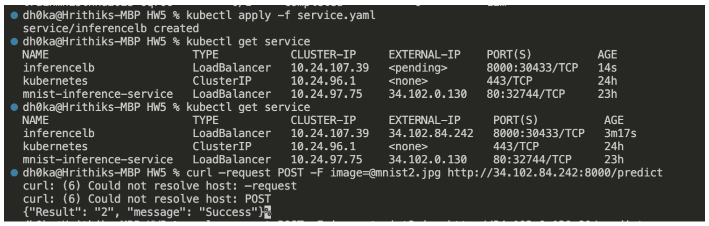
2) Then tag the image by docker tag mnist_inference_hhd2023 akohd/mnist_inference_hhd2023.
3) Then push the inference image to the dockerhub registry by using the command docker push akohd/mnist_inference_hhd2023.
4) Now, similar to training, we will create infer.yaml on our cloud shell using vim infer.yaml.
5) Deploy the inference application: kubectl apply -f infer.yaml
6) To verify whether the deployment is successful or not we can check kubectl get deployments and kubectl get pods. The status running tells that the deployment is up and can be consumed by the service.



```
● dh0ka@Hrithiks-MBP HW5 % kubectl apply -f infer.yaml
  deployment.apps/inferjob created
● dh0ka@Hrithiks-MBP HW5 % kubectl get deployments
  NAME           READY   UP-TO-DATE   AVAILABLE   AGE
  inferencejob   1/1     1            1           23h
  inferjob       1/1     1            1           14s
```

E. Creating Service on Kubernetes

1) Since our deployment is ready, we will now create a service.yaml.
2) Deploy the service: kubectl apply -f service.yaml.
3) Verify the service is running properly by kubectl get service.
4) Describe the service using kubectl describe service mnist-inference-service.
5) Now we can see our service is up and running.The external ip is the ip exposed to the end user for interacting with the application.



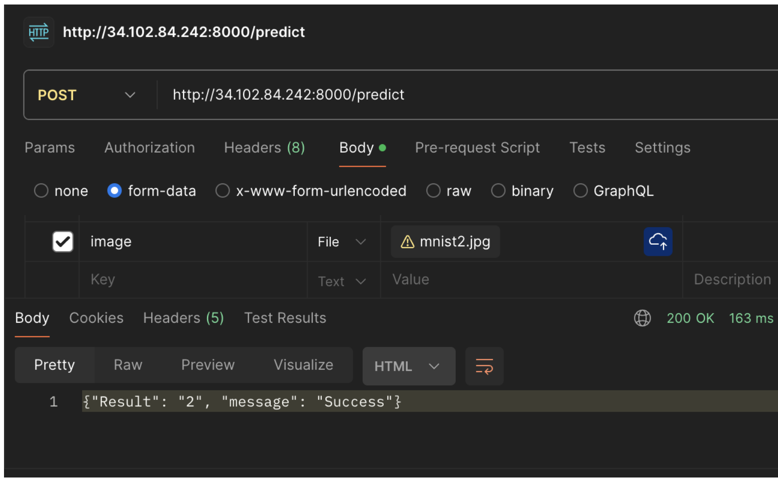
```
● dh0ka@Hrithiks-MBP HW5 % kubectl apply -f service.yaml
  service/inferencelb created
● dh0ka@Hrithiks-MBP HW5 % kubectl get service
  NAME                      TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)          AGE
  inferencelb               LoadBalancer   10.24.107.39   <pending>      8000:30433/TCP   14s
  kubernetes                ClusterIP      10.24.96.1     <none>         443/TCP          24h
  mnist-inference-service   LoadBalancer   10.24.97.75    34.102.0.130   80:32744/TCP     23h
● dh0ka@Hrithiks-MBP HW5 % kubectl get service
  NAME                      TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)          AGE
  inferencelb               LoadBalancer   10.24.107.39   34.102.84.242   8000:30433/TCP   3m17s
  kubernetes                ClusterIP      10.24.96.1     <none>          443/TCP          24h
  mnist-inference-service   LoadBalancer   10.24.97.75    34.102.0.130    80:32744/TCP     23h
● dh0ka@Hrithiks-MBP HW5 % curl -request POST -F image=@mnist2.jpg http://34.102.84.242:8000/predict
  curl: (6) Could not resolve host: -request
  curl: (6) Could not resolve host: POST
  {"Result": "2", "message": "Success"}%
```

III. Observation

After the external ip is up and running, from commandline we use the curl command to hit the inference Flask server with an image to predict: curl –request POST -F image=@mnist2.jpg http://34.102.84.242:8000/predict. This returns a json object containing the predicted number and a message displaying success or failure depending on the input.

The API can also be tested using the Postman application to check if the external api can be accessed and returns the desired result. You can follow the steps to below for testing on postman:

1) Create a new POST request
2) Enter the external ip with the correct api route
3) For body, the method should accept 'form-data', with the input being a file having a key named 'image'.
4) Click on send, to send the request to the server and process the image
5) After successful execution, we get a json object containing the approriate response based on the input.



http://34.102.84.242:8000/predict

POST http://34.102.84.242:8000/predict

Body: form-data

	Key		Value
✓	image	File	mnist2.jpg

Body — 200 OK 163 ms

```
{"Result": "2", "message": "Success"}
```

IV. Conclusion

This paper demonstrated the feasibility of using a Kubernetes cluster for training, deploying, and performing inference on the MNIST handwritten digit classification dataset. We successfully containerized the training and inference processes, leveraging the scalability and resource management capabilities of Kubernetes. The results highlight the effectiveness of this approach, showcasing distributed training for faster model development and seamless deployment for real-time predictions. Additionally, Kubernetes facilitates efficient resource utilization by dynamically allocating resources based on workload demands.

This work paves the way for exploring more complex deep learning models and real-world datasets within a Kubernetes environment. Future research directions include investigating advanced scheduling strategies for optimizing training job placement and resource allocation, as well as integrating with container orchestration tools for a more robust and automated machine learning pipeline.

References

[1] Blog of deploying an ML model on Docker. https://towardsdatascience.com/build-and-run-a-docker-container-for-your-machine-learning-model-60209c2d7a7f
[2] MNIST code repo=¿ https://github.com/pytorch/examples/tree/main/mnist_rnn
[3] Containzerization wikipedia=¿ https://en.wikipedia.org/wiki/Containerization_(computin
[4] Kubernetes Wikipedia =¿ https://en.wikipedia.org/wiki/Kubernetes
[5] Kubernetes official documentation =¿ https://kubernetes.io/docs/home/
[6] GKE official documentation =¿ https://cloud.google.com/kubernetes-engine?hl=en
[7] Medium blog of deploying containers to kubernetes =¿ https://tsai-liming.medium.com/part-3-deploying-your-data-science-containers-to-kubernetes-aaae769144ec
[8] Gradio documentation =¿ https://www.gradio.app/docs/interface