Cloud and machine learning - homework5

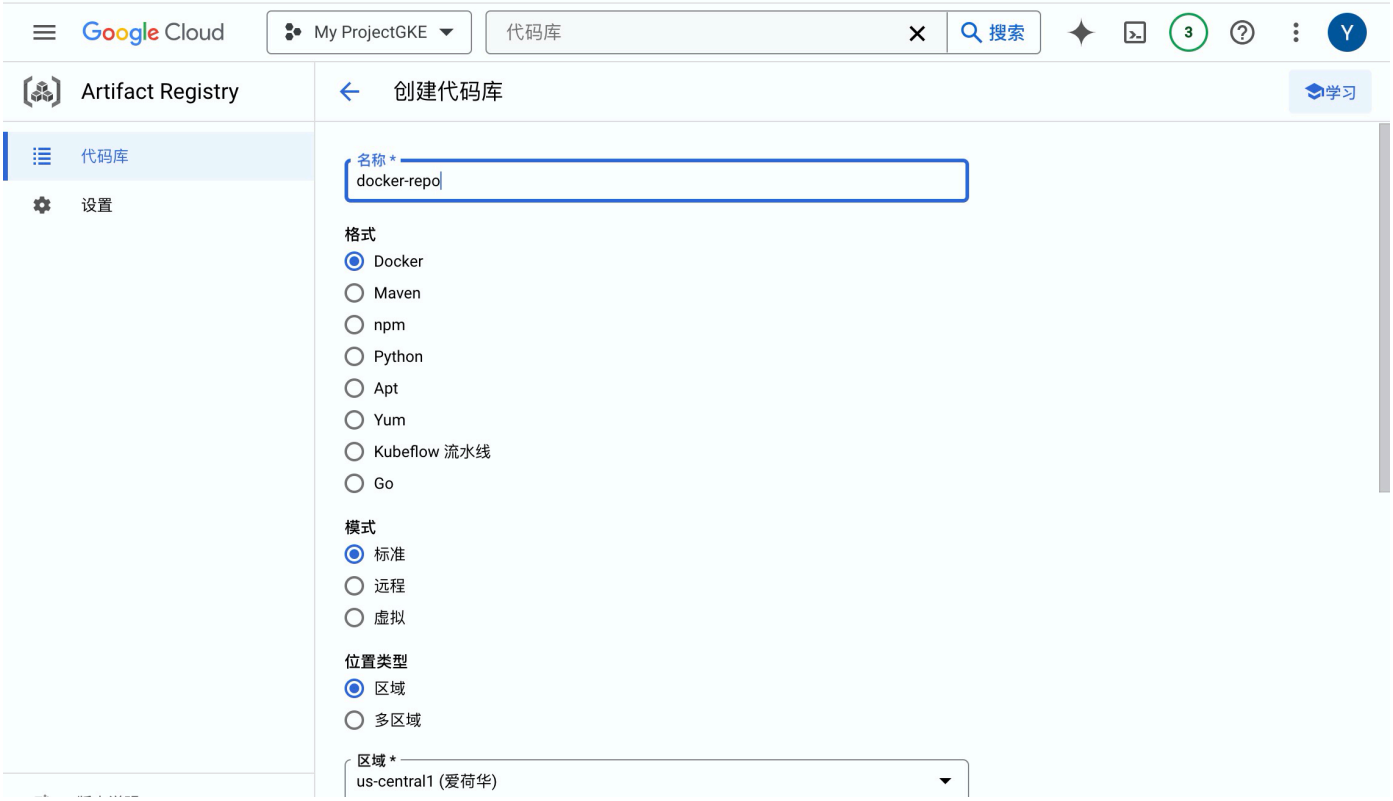
Environment Setup and Workflow Documentation

1. Installation of kubectl

The Kubernetes command-line tool, `kubectl`, is essential for managing Kubernetes clusters. It allows for the deployment and management of applications, inspection and management of cluster resources, and viewing of logs.

2. Enabling the Artifact Registry API

Enabling the Artifact Registry API is crucial for storing Docker images on Google Cloud, facilitating the management and deployment of containerized applications.



3. Dockerfile Creation and Image Building and Push to Artifact Registry

A Dockerfile was created to define the environment and specifications needed for the MNIST training model. The Docker image was built using the command:

```
docker build -t us-central1-docker.pkg.dev/my-projectgke/docker-repo/mnist-
training:latest .
docker tag us-central1-docker.pkg.dev/my-projectgke/inference:latest us-central1-
docker.pkg.dev/my-projectgke/docker-repo/inference:latest
docker push us-central1-docker.pkg.dev/my-projectgke/docker-repo/inference:latest
```

```
gongyitong@gongyitongMBP DL_workflow % ls
train.py
gongyitong@gongyitongMBP DL_workflow % vim Dockerfile
gongyitong@gongyitongMBP DL_workflow % docker build -t my-app:v1 .
[+] Building 185.6s (9/9) FINISHED
 => [internal] load build definition from Dockerfile                                                                                                0.0s
 => => transferring dockerfile: 297B                                                                                                                0.0s
 => [internal] load .dockerignore                                                                                                                   0.0s
 => => transferring context: 2B                                                                                                                     0.0s
 => [internal] load metadata for docker.io/library/python:3.8                                                                                       1.2s
 => [1/4] FROM docker.io/library/python:3.8@sha256:6ea16099cac9f66419d1fc3a63aaa9d783214e8e38d2a1c0db2bfb0852ef9b7d                                61.1s
 => => resolve docker.io/library/python:3.8@sha256:6ea16099cac9f66419d1fc3a63aaa9d783214e8e38d2a1c0db2bfb0852ef9b7d                                  0.0s
 => => sha256:3b74f020507cc447d399db5c64cae14579323514dcc2c1d4fb2256fdfd304b87 2.01kB / 2.01kB                                                      0.0s
 => => sha256:1468e7ff95fcb865fbc4dee7094f8b99c4dcddd6eb2180cf044c7396baf6fc2f 49.58MB / 49.58MB                                                   32.5s
 => => sha256:2cf9c2b42f41b1845f3e4421b723d56146db82939dc884555e07776e18132f4 24.05MB / 24.05MB                                                    9.3s
 => => sha256:6ea16099cac9f66419d1fc3a63aaa9d783214e8e38d2a1c0db2bfb0852ef9b7d 1.86kB / 1.86kB                                                      0.0s
 => => sha256:17b8b8dc7004de4f94b101a88b71bffd77132e07fd5ff3a59299c6333d2bf7e5 7.36kB / 7.36kB                                                      0.0s
 => => sha256:c4c40c3e3cdf945721f480e1d939aac857876fdb5c33b8fbfcf655c63b0b9428 64.14MB / 64.14MB                                                    13.7s
 => => sha256:c05cc1123d7e335d59b0f465c23b7ad2ad27f4875b6c3eab41c65a9b50efa382 211.18MB / 211.18MB                                                  50.2s
 => => sha256:b6f29ccdcc551647511d3473f89c94b2ee7fbce3e65226908ea74cfc5c586697 6.39MB / 6.39MB                                                      20.6s
 => => sha256:f4a2192a24b40976b6a86429e39af3b33ceddcfa42c1a1e32fe38c2a0ebf0de9 15.20MB / 15.20MB                                                    22.6s
 => => sha256:6310fd3a3471e01c27cefdd6a9ef5ff29e76ddc52ddb173e85cdc715f17d6209 243B / 243B                                                          22.7s
 => => sha256:f21b2256fb6a6f827ea09aae5e810926d87d1a0691efe9d70e2d0a95a7470a28 2.85MB / 2.85MB                                                      23.5s
 => => extracting sha256:1468e7ff95fcb865fbc4dee7094f8b99c4dcddd6eb2180cf044c7396baf6fc2f                                                            2.3s
 => => extracting sha256:2cf9c2b42f41b1845f3e4421b723d56146db82939dc884555e07776e18132f4                                                            0.9s
 => => extracting sha256:c4c40c3e3cdf945721f480e1d939aac857876fdb5c33b8fbfcf655c63b0b9428                                                            3.7s
 => => extracting sha256:c05cc1123d7e335d59b0f465c23b7ad2ad27f4875b6c3eab41c65a9b50efa382                                                            8.7s
 => => extracting sha256:b6f29ccdcc551647511d3473f89c94b2ee7fbce3e65226908ea74cfc5c586697                                                            0.3s
 => => extracting sha256:f4a2192a24b40976b6a86429e39af3b33ceddcfa42c1a1e32fe38c2a0ebf0de9                                                            0.7s
 => => extracting sha256:6310fd3a3471e01c27cefdd6a9ef5ff29e76ddc52ddb173e85cdc715f17d6209                                                            0.0s
 => => extracting sha256:f21b2256fb6a6f827ea09aae5e810926d87d1a0691efe9d70e2d0a95a7470a28                                                            0.3s
 => [internal] load build context                                                                                                                   0.0s
 => => transferring context: 22.16kB                                                                                                                0.0s
 => [2/4] WORKDIR /app                                                                                                                              0.4s
 => [3/4] COPY . /app                                                                                                                               0.0s
 => [4/4] RUN pip install tensorflow                                                                                                              108.5s
 => exporting to image                                                                                                                             14.2s
 => => exporting layers                                                                                                                            14.1s
 => => writing image sha256:44190569b3daf4c4a9a74477515754947a3ad1ba817e242e59227d15d7598aa6                                                        0.0s
 => => naming to docker.io/library/my-app:v1                                                                                                        0.0s

gongyitong@gongyitongMBP DL_workflow % gcloud auth configure-docker us-central1-docker.pkg.dev
Adding credentials for: us-central1-docker.pkg.dev
After update, the following will be written to your Docker config file located at [/Users/gongyitong/.docker/config.json]:
 {
  "credHelpers": {
    "us-central1-docker.pkg.dev": "gcloud"
  }
}

Do you want to continue (Y/n)?  Y

Docker configuration file updated.
```

```
gongyitong@gongyitongMBP DL_workflow % docker build -t us-central1-docker.pkg.dev/my-projectgke/docker-repo/mnist-training:latest .

[+] Building 121.8s (9/9) FINISHED
 => [internal] load build definition from Dockerfile                                                                                                0.0s
 => => transferring dockerfile: 297B                                                                                                                0.0s
apiVersion: batch/v1
 => [internal] load .dockerignore                                                                                                                   0.0s
 => => transferring context: 2B                                                                                                                     0.0s
 => [internal] load metadata for docker.io/library/python:3.8                                                                                       0.8s
 => [1/4] FROM docker.io/library/python:3.8@sha256:6ea16099cac9f66419d1fc3a63aaa9d783214e8e38d2a1c0db2bfb0852ef9b7d                                 0.0s
 => [internal] load build context                                                                                                                   0.0s
 => => transferring context: 56.82kB                                                                                                                0.0s
 => CACHED [2/4] WORKDIR /app                                                                                                                       0.0s
 => [3/4] COPY . /app                                                                                                                               0.0s
 => [4/4] RUN pip install tensorflow                                                                                                              106.8s
 => exporting to image                                                                                                                             14.0s
 => => exporting layers                                                                                                                            13.9s
 => => writing image sha256:baf15145ad3d68d4b5912cc685b1b17c96d397846ccd84a07f4e99bd1df245ff                                                        0.0s
 => => naming to us-central1-docker.pkg.dev/my-projectgke/docker-repo/mnist-training:latest                                                         0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
gongyitong@gongyitongMBP DL_workflow % docker push us-central1-docker.pkg.dev/my-projectgke/docker-repo/mnist-training:latest
The push refers to repository [us-central1-docker.pkg.dev/my-projectgke/docker-repo/mnist-training]
6a2361a9a5ae: Pushed
8d22864d5714: Pushed
8624a871870c: Layer already exists
41edac4edf75: Layer already exists
1dde678d2269: Layer already exists
3a1b6e184b4a: Layer already exists
89ca33c95b2e: Layer already exists
apiVersion: batch/v1
83db175c22e2: Layer already exists
c5d13b2949a2: Layer already exists
7e43f593c900: Layer already exists
072686bcd3db: Layer already exists
latest: digest: sha256:6d3ae12b2ca72e010996a402ee6bd6bedc70e8f84d251b381124e5f2ed464c10 size: 2637
```

```
gongyitong@gongyitongMBP inference % docker tag us-central1-docker.pkg.dev/my-projectgke/inference:latest us-central1-docker.pkg.dev/my-projectgke/docker-repo/inference:latest
gongyitong@gongyitongMBP inference % docker push us-central1-docker.pkg.dev/my-projectgke/docker-repo/inference:latest
The push refers to repository [us-central1-docker.pkg.dev/my-projectgke/docker-repo/inference]
6353dbe70079: Pushed
e2a2174822dd: Pushed
fae83ac67d20: Pushing [>                                                  ]  29.43MB/2.123GB
b60a1f471434: Pushed
8d8e7f754ef8: Pushed
cbb0bec46633: Pushed
7a75d57a5024: Pushed
52ec5a4316fa: Pushing [====================>                              ]  35.78MB/74.83MB
```

projectid:my-projectgke

remotecode：docker-repo

Image: mnist-training(for model training), inference(for apply model)

4. Persistent Volume Configuration

To ensure data persistence across pod restarts and failures, a persistent volume was configured. This allows for the storage of model data externally from the pod lifecycle.

```
gongyitong@gongyitongMBP DL_workflow % vim training-job.yaml
gongyitong@gongyitongMBP DL_workflow % kubectl apply -f training-job.yaml
job.batch/mnist-training-job created
gongyitong@gongyitongMBP DL_workflow %
```

```
gongyitong@gongyitongMBP DL_workflow % kubectl get pvc
kubectl get jobs
kubectl describe job mnist-training-job

NAME              STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
mnist-model-pvc   Bound    pvc-331e2181-2eed-4ae9-ac6e-4a006ec94a83   2Gi        RWO            standard       28m
NAME                COMPLETIONS   DURATION   AGE
mnist-training-job   0/1           24m        24m
Name:           mnist-training-job
Namespace:      default
Selector:       batch.kubernetes.io/controller-uid=696bb88d-9f96-4974-bb89-cec668aa280a
Labels:         batch.kubernetes.io/controller-uid=696bb88d-9f96-4974-bb89-cec668aa280a
                batch.kubernetes.io/job-name=mnist-training-job
                controller-uid=696bb88d-9f96-4974-bb89-cec668aa280a
                job-name=mnist-training-job
Annotations:    <none>
Parallelism:    1
Completions:    1
Start Time:     Mon, 29 Apr 2024 08:57:00 -0400
Pods Statuses:  1 Running / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  batch.kubernetes.io/controller-uid=696bb88d-9f96-4974-bb89-cec668aa280a
           batch.kubernetes.io/job-name=mnist-training-job
           controller-uid=696bb88d-9f96-4974-bb89-cec668aa280a
           job-name=mnist-training-job
  Containers:
   mnist-training:
    Image:      us-central1-docker.pkg.dev/my-projectgke/docker-repo/my-app:latest
    Port:       <none>
    Host Port:  <none>
    Command:
      python
      train.py
    Environment:  <none>
    Mounts:
      /model from model-volume (rw)
  Volumes:
   model-volume:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  mnist-model-pvc
    ReadOnly:   false
Events:
  Type    Reason            Age   From            Message
  ----    ------            ----  ----            -------
  Normal  SuccessfulCreate  24m   job-controller  Created pod: mnist-training-job-8brlh
gongyitong@gongyitongMBP DL_workflow %
```

modify the model is Persistented：

```
gongyitong@gongyitongMBP DL_workflow % kubectl exec -it temp-pod -- sh

/ #
/ # ls
bin    dev    etc    home   lib    lib64  model  proc   root   sys    tmp    usr    var
/ # cd model
/model # ls
lost+found      mnist_model.h5
/model # cd ../
/ # exit
```

5. Inference Service Setup

An inference image was created and pushed to the Artifact Registry. Kubernetes deployments and services were configured to manage the lifecycle of the inference pods and expose them externally:

```
kubectl apply -f deployment.yaml

kubectl apply -f service.yaml
kubectl get pods //check the status of the service
```

```
gongyitong@gongyitongMBP inference % kubectl get pods
NAME                                    READY   STATUS      RESTARTS   AGE
inference-deployment-64ccbd77cf-5pvdf   1/1     Running     0          11s
mnist-training-job-ps4xc                0/1     Completed   0          9m39s
nginx-1-584cd97758-hbxmd                1/1     Running     0          22h
nginx-1-584cd97758-ldst4                1/1     Running     0          22h
nginx-1-584cd97758-rsxzp                1/1     Running     0          22h
```

```
kubectl get services //check the external_ip of service
```

```
gongyitong@gongyitongMBP inference % kubectl get services
NAME               TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
inference-service  LoadBalancer   10.98.218.54   34.69.143.226   80:30955/TCP   119m
kubernetes         ClusterIP      10.98.208.1    <none>          443/TCP        22h
```

6. Interacting with the Model

A Python script, `send_inference_request.py`, was used to send a POST request to the inference service. This script processes an image and sends it to the model for inference, demonstrating the model's ability to predict based on input data.

```python
def send_request(image_data, url):
    headers = {'Content-Type': 'application/json'}
    payload = json.dumps({"image": image_data})
    response = requests.post(url, data=payload, headers=headers)
    return response.json()

if __name__ == "__main__":
    image_path = '1.jpg'  # path to the image
    inference_url = 'http://34.69.143.226/predict'  # url for the outer ip
image_data = prepare_image(image_path)

result = send_request(image_data, inference_url)
print("Inference result:", result)
```

Get the training result:

```
gongyitong@gongyitongMBP inference % python3 send_inference_request.py
Inference result: {'prediction': [[2.930822300684630 4e-15, 9.230327302925087e-18, 1.5893772842973108e-12, 5.861031127096794e-08, 3.563873731234626e-36, 1.0, 2.092344526805067 8e-11, 8.733351819500967e-10, 4.1031620629802075e-16, 3.080534103562407e-15]]}
```

Experiences and Observations

Kubernetes Controllers Used

For both training and inference, Kubernetes Deployments were utilized. Deployments provide a declarative method for updating applications and scaling. They are suitable for stateless applications and were ideal for managing the lifecycle of both training and inference components.

Why Kubernetes and GKE?

Kubernetes offers robust orchestration capabilities, including automated deployment, scaling, and management of containerized applications. Google Kubernetes Engine (GKE) enhances this with managed services, simplifying cluster maintenance and providing a scalable infrastructure that can adapt to changing loads, crucial for machine learning workflows.

Challenges and Learnings

The primary challenges faced involved managing the dependencies and environments within the containers, ensuring that the applications were isolated and reproducible. Additionally, configuring network policies and persistent volumes required careful consideration to ensure data security and integrity.

This experience highlighted the benefits of using containerized environments for machine learning workflows, particularly the flexibility in managing and deploying applications. It also demonstrated the effectiveness of Kubernetes and cloud services in handling scalable, distributed systems.