Building and Training a Machine Learning Model, Containerizing, and Deploying a Machine Learning Model Using Flask, Docker Images, and Kubernetes Services

This project demonstrates the end-to-end process of building, training, and deploying a machine learning model within a robust DevOps environment. The workflow involves:

1. **Building and Training the Machine Learning Model**:
    - A simple machine learning model is created using Python and libraries like scikit-learn.
    - The model is trained with sample data, serialized into a model.pkl file, and saved for deployment.
2. **Creating a Flask API for Model Serving**:
    - A Flask application is developed to serve the trained model, providing endpoints for making predictions.
    - The API processes input data, loads the serialized model, and returns predictions in real-time.
3. **Containerizing the Application Using Docker**:
    - A Docker image is created for the Flask application, bundling the app code, dependencies, and the trained model.
    - Docker commands are used to build, test, and run the container locally to ensure functionality.
4. **Deploying the Containerized Application Using Kubernetes**:
    - Minikube is used to simulate a Kubernetes cluster locally.
    - The Docker image is deployed as a Kubernetes Pod, with a Service created to expose the application.
    - kubectl commands manage the deployment, scaling, and health of the Pods.

Divya Murugan

Step 1: Create the ML Model

1. **Directory structure**:

ml-app/

├── app/

│ ├── model.py

│ ├── app.py

│ └── requirements.txt

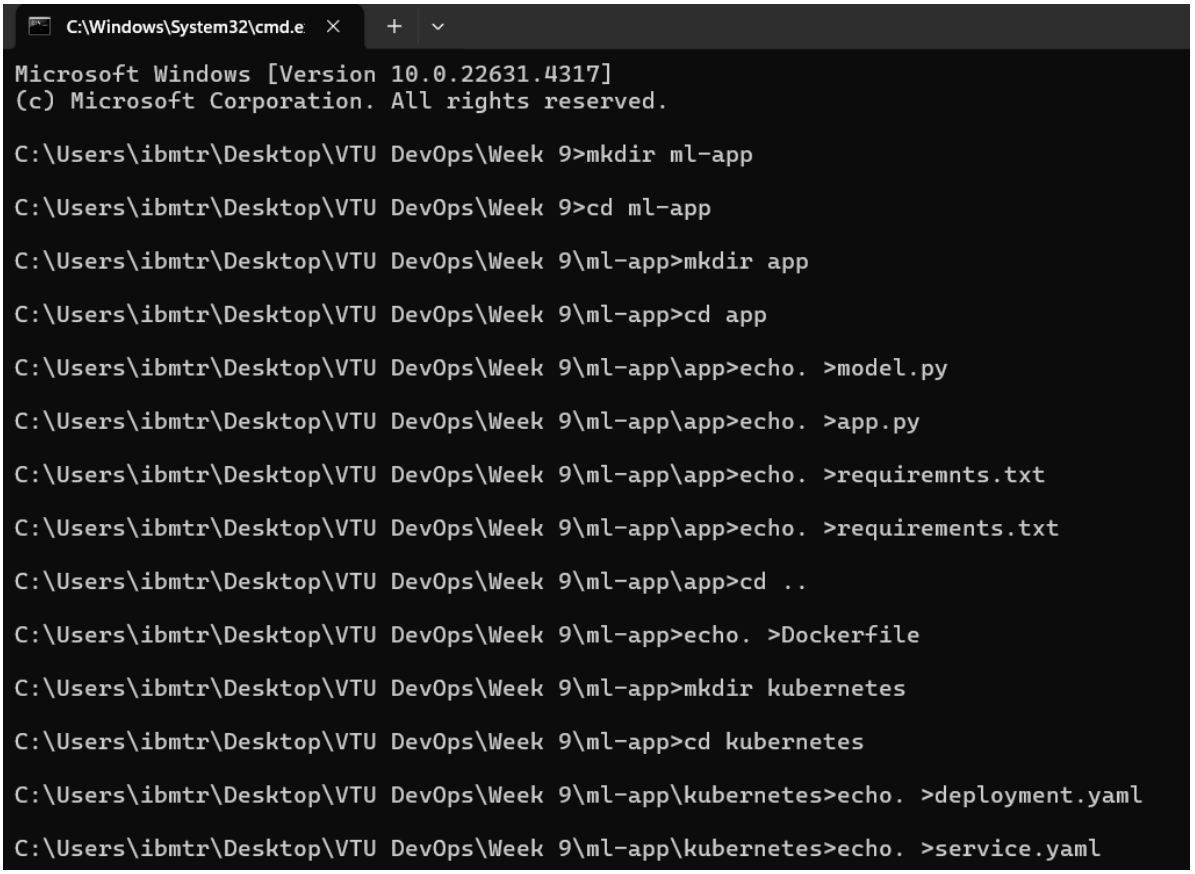
├── Dockerfile

├── kubernetes/

│ ├── deployment.yaml

│ └── service.yaml

└── README.md



2. **ML model code (model.py)**:

```
import pickle
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
# Train and save a model
def train_model():
    try:
        # Load dataset
        data = load_iris()
        X, y = data.data, data.target
        # Train a simple model
        model = RandomForestClassifier()
        model.fit(X, y)
        # Save the model to a file
        with open("model.pkl", "wb") as file:
            pickle.dump(model, file)
        print("Model created and saved as 'model.pkl'")
    except Exception as e:
        print(f"Error: {e}")
if __name__== "__main__":
    train_model()
```

3. **Flask app code (app.py)**:

```
import pickle
from flask import Flask, request, jsonify
app = Flask(__name__)
# Load the trained model
with open("model.pkl", "rb") as file:
    model = pickle.load(file)


@app.route("/")
def home():
    return "Welcome to the ML App!"


@app.route("/predict", methods=["POST"])
def predict():
    data = request.json
    prediction = model.predict([data["features"]])
    return jsonify({"prediction": prediction.tolist()})


if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

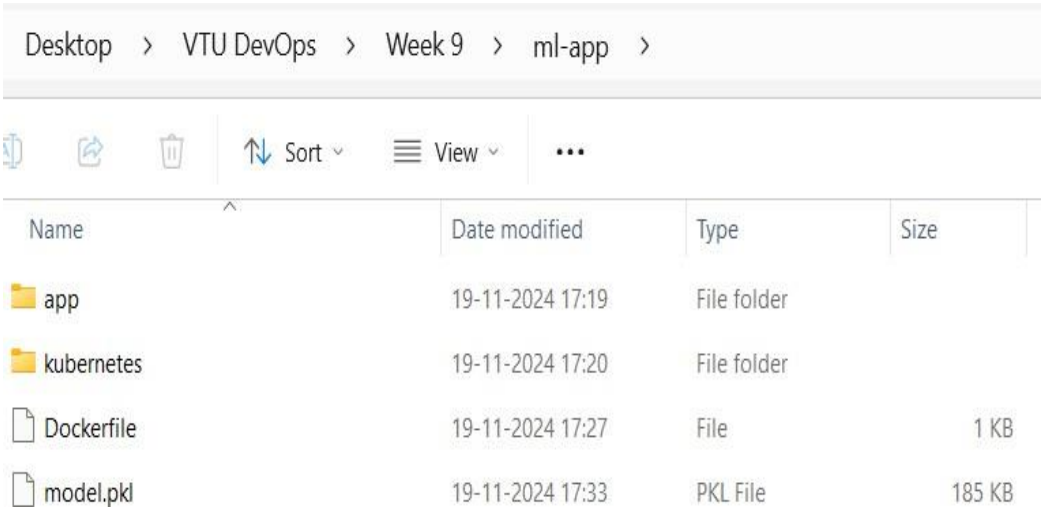
4. **Requirements file (requirements.txt)**:

```
Flask==2.3.3
scikit-learn==1.3.2
```

5. **Train the model**: Run the model.py script to generate the model.pkl file:

python app/model.py

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\ml-app>python app/model.py
Model created and saved as 'model.pkl'
```



Step 2: Create a Docker Image

1. **Dockerfile**:

```
FROM python:3.9-slim

WORKDIR /app

# Copy the application and the model file into the Docker image

COPY app/ /app/

COPY model.pkl /app/

RUN pip install -r requirements.txt

EXPOSE 5000

CMD ["python", "app.py"]
```

Step 3: Verify Image Availability

Ensure Kubernetes can access the image:

docker images

2. **Build the Docker image**:

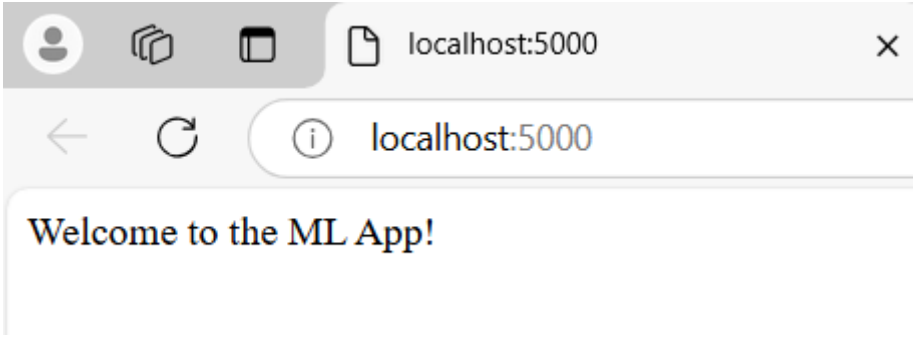
docker build -t ml-app .

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\ml-app>docker build -t ml-app .
[+] Building 4.3s (11/11) FINISHED                                                                  docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                0.2s
 => => transferring dockerfile: 263B                                                                                0.1s
 => [internal] load metadata for docker.io/library/python:3.9-slim                                                  3.5s
 => [auth] library/python:pull token for registry-1.docker.io                                                       0.0s
 => [internal] load .dockerignore                                                                                   0.0s
 => => transferring context: 2B                                                                                     0.0s
 => [1/5] FROM docker.io/library/python:3.9-slim@sha256:6250eb7983c08b3cf5a7db9309f8630d3ca03dd152158fa37a3f8daaf   0.0s
 => [internal] load build context                                                                                   0.0s
 => => transferring context: 156B                                                                                   0.0s
 => CACHED [2/5] WORKDIR /app                                                                                       0.0s
 => CACHED [3/5] COPY app/ /app/                                                                                    0.0s
 => CACHED [4/5] COPY model.pkl /app/                                                                               0.0s
 => CACHED [5/5] RUN pip install -r requirements.txt                                                                0.0s
 => exporting to image                                                                                              0.0s
 => => exporting layers                                                                                             0.0s
 => => writing image sha256:b03c2e82a8aa621c8baf41a6b7feb4a15aaa2f814ff1c8b3b4e92f8c7c998aa8                        0.0s
 => => naming to docker.io/library/ml-app                                                                           0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/hqjai3kbmyhasej12wj50q1yo

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview

C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\ml-app>
```

3. **Test the Docker container locally**:

docker run -p 5000:5000 ml-app

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\ml-app>docker run -p 5000:5000 ml-app
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
172.17.0.1 - - [19/Nov/2024 12:07:09] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [19/Nov/2024 12:07:09] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [19/Nov/2024 12:07:11] "GET /favicon.ico HTTP/1.1" 404 -
172.17.0.1 - - [19/Nov/2024 12:07:11] "GET /favicon.ico HTTP/1.1" 404 -
```

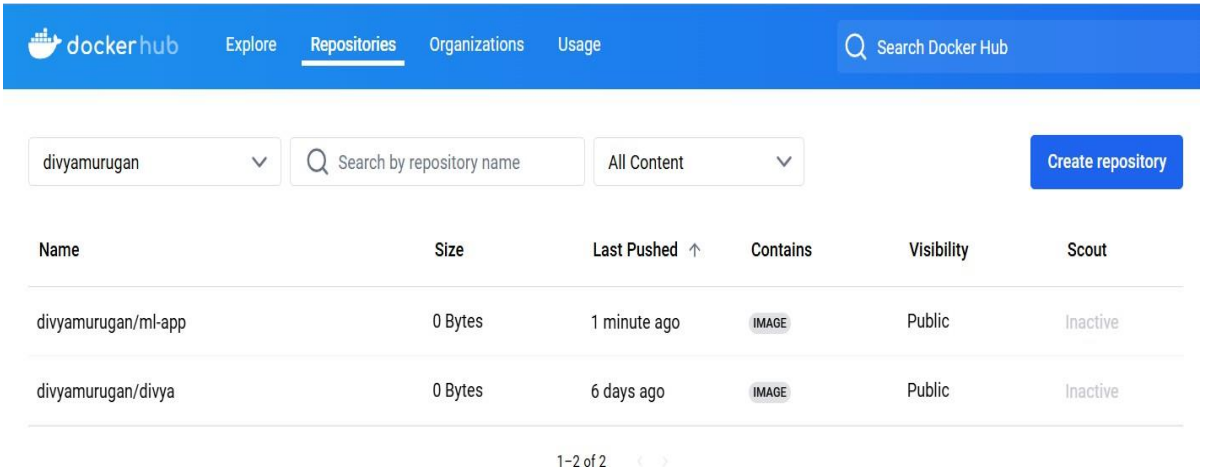


If you’re using a remote image, ensure it is pushed to a container registry like Docker Hub:

docker tag ml-app:latest <your-dockerhub-username>/ml-app:latest

docker push <your-dockerhub-username>/ml-app:latest

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\ml-app>docker tag ml-app:latest divyamurugan/ml-app:latest

C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\ml-app>docker push divyamurugan/ml-app:latest
The push refers to repository [docker.io/divyamurugan/ml-app]
31be0e4c7d7e: Pushed
22af955e1935: Pushed
0af63ec193bc: Pushed
0cbe6fec9a4d: Pushed
aacba17e24d9: Mounted from library/python
f751ad7c65c4: Mounted from library/python
7822e749b484: Mounted from library/python
c3548211b826: Mounted from library/python
latest: digest: sha256:4b597fd1c87876d44ca339720f4fc9ce7cd7b49847f13bfa510b545be88908ef size: 1994
```



Step 3: Deploy with Minikube and kubectl

1. **Start Minikube**:

minikube start

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\ml-app>minikube start
* minikube v1.34.0 on Microsoft Windows 11 Home Single Language 10.0.22631.4317 Build 22631.4317
* Using the docker driver based on existing profile
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.45 ...
* Restarting existing docker container for "minikube" ...
! Failing to connect to https://registry.k8s.io/ from inside the minikube container
* To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking
/proxy/
* Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: default-storageclass, storage-provisioner
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\ml-app>minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

2. **Create Kubernetes manifests**:

   - **Deployment (deployment.yaml)**:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ml-app
spec:
  replicas: 1
  selector:
  matchLabels:
    app: ml-app
```

```
  template:
    metadata:
      labels:
        app: ml-app
    spec:
      containers:
      - name: ml-app
        image: divyamurugan/ml-app:latest
        ports:
        - containerPort: 5000
```

- Update the deployment.yaml to reference the correct image:

  image: <your-dockerhub-username>/ml-app:latest

- **Service (service.yaml)**:

```
apiVersion: v1
kind: Service
metadata:
  name: ml-app-service
spec:
  selector:
    app: ml-app
  ports:
  - protocol: TCP
    port: 80
```

```
  targetPort: 5000

type: NodePort
```

3. **Apply the manifests**:

kubectl apply -f kubernetes/deployment.yaml

kubectl apply -f kubernetes/service.yaml

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\ml-app>kubectl apply -f kubernetes/deployment.yaml
deployment.apps/ml-app created

C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\ml-app>kubectl apply -f kubernetes/service.yaml
service/ml-app-service created
```

Check Deployment and Service

1. **Check Deployment**: Verify the status of the Deployment:

kubectl get deployments

kubectl get pods

kubectl get svc

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\ml-app>kubectl get deployments
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
ml-app   1/1     1            1           4m5s

C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\ml-app>kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
ml-app-645c8c678d-nv57g   1/1     Running   0          4m10s

C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\ml-app>kubectl get svc
NAME             TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes       ClusterIP   10.96.0.1        <none>        443/TCP        5m20s
ml-app-service   NodePort    10.102.124.192   <none>        80:30638/TCP   4m3s
```

4. **Access the service**: Get the service URL using Minikube:

minikube service ml-app-service

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\ml-app>minikube service ml-app-service
|-----------|---------------|-------------|-------------------------|
| NAMESPACE |      NAME     | TARGET PORT |           URL           |
|-----------|---------------|-------------|-------------------------|
| default   | ml-app-service |          80 | http://192.168.49.2:30638 |
|-----------|---------------|-------------|-------------------------|
* Starting tunnel for service ml-app-service.
|-----------|---------------|-------------|-----------------------|
| NAMESPACE |      NAME     | TARGET PORT |          URL          |
|-----------|---------------|-------------|-----------------------|
| default   | ml-app-service |             | http://127.0.0.1:50799 |
|-----------|---------------|-------------|-----------------------|
* Opening service default/ml-app-service in default browser...
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```