Docker & Kubernetes Demo - Reference

Copy over from the fastapi/ folder the fastapi app and the model - joblib file

```shell
# one time addition of the user to docker group
# allows user to run Docker commands without needing to superuser permissions
sudo usermod -aG docker $USER

# build docker image with the named tag iris-api-v2
docker build -t iris-api .

# docker run to create docker container from docker image
# will return container ID (e.g. bKe3451hs)
docker run -d -p 8200:8200 iris-api

# -d is detached mode
# 8200 of local app within docker is mapped to 8200 of the host machine
# ensure 8200 port is open in the vpc firewall
# can also use `-name <CONTAINER_NAME>` for naming the container
# check the curl command to check the deployment
# curl -X POST "<DEPLOYMENT_IP>/predict/" -H "Content-Type: application/json"
-d "("sepal_length": 5.1, "sepal width": 3.5, "petal_length": 1.4,
"petal_width": 0.2)
# can check deployment logs via `docker logs <CONTAINER_ID>` to check container
logs
```

To take the docker container down:

```shell
# list containers
docker ps
# then stop the container using
docker stop <CONTAINER_ID>
```

Dockerfile

```Shell
#1. Use official Python base image
FROM python:3.10-slim
#2. Set working directory
WORKDIR /app
#3. Copy files
COPY /app
#4. Install dependencies
RUN pip install --no-cache-dir -r requirements.txt
#5. Expose port
EXPOSE 8200
#6. Command to run the server
CMD ["uvicorn", "iris_fastapi:app", "--host", "0.0.0.0", "--port", "8200"]
```

requirements.txt

```Python
fastapi
uvicorn
scikit-learn
joblib
numpy
```

iris_fastapi.py

```Python
# iris_fastapi.py
from fastapi import FastAPI
from pydantic import BaseModel
import joblib
import numpy as np
import pandas as pd
```

```
app = FastAPI(title="Iris Classifier API")

# Load model
model = joblib.load("model.joblib")

# Input schema
class IrisInput(BaseModel):
    sepal_length: float
    sepal_width: float
    petal_length: float
    petal_width: float

@app.get("/")
def read_root():
    return {"message": "Welcome to the Iris Classifier API!"}

@app.post("/predict/")
def predict_species(data: IrisInput):
    input_df = pd.DataFrame([data.dict()])
    prediction = model.predict(input_df)[0]
    return {
        "predicted_class": prediction
    }
```

To deploy to kubernetes

Shell

```
# Enable k8s engine API on cloud console

gcloud services enable artifactregistry.googleapis.com

gcloud artifacts repositories create my-repo |
--repository-format=docker |
--location=us-central1 \
--description="Docker repo for ML models"

gcloud auth configure-docker us-central1-docker.pkg.dev

# to get the image name corresponding to the model you created.
docker images
```

```
# Tag the image

docker tag iris-api
us-central1-docker.pkg.dev/dulcet-bastion-452612-v4/my-repo/iris-api:latest

# Push the image - docker push

us-central1-docker.pkg.dev/dulcet-bastion-452612-v4/my-repo/iris-api:latest

# Artifact registry is now populated. From now on, even if the k8s cluster that
will get created is down or deleted, we can recreate using the image in the AR.

# In GKE:
# Create a cluster test-iris-v1, iris-classifier namespace
# Deploy - takes about 5 mins
# Click on "Connect" -> "Open workloads dashboard"
# Then do a deploy by selecting the existing image
# Then expose it using a Load Balancer

# Optional : For troubleshooting
# Get cluster credentials locally - (issued from cloud shell) - gcloud

container clusters get-credentials test-iris-v1-cluster --zone us-central1
--project dulcet-bastion-452612-v4

# get kubernetes deployed pods
kubectl get pods

# get kubernetes service
kubectl get service

# to edit kubernetes service
kubectl edit service <your service name>

# Locate the spec.ports
# Change target port to 8200
```