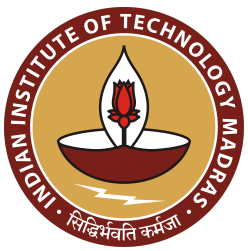
Big Data Laboratory

Assignment 7
Deployment of ML model using Fast Api and monitoring

by

Anik Bhowmick
AE20B102

Instructor: Sudarshan
Teaching Assistant: Jashaswimalya Acharjee



Contents

1

Introduction

As part of MLOps, this assignment aims to deploy the ML model on a web server using Fast Api and monitor it with Prometheus and Grafana. FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.8+ based on standard Python type hints. As a part of this assignment, we had to perform versioning of our project using Git Hub. To have GitHub access to all the files **click here**.

1.1. The problem statement

We have to set up a web interface to deploy an ML model trained on digit classification tasks. FastAPI has to be used to create the API, and Swagger UI for the webserver. On top of that, we have to use Prometheus and Grafana for server health monitoring. The monitoring metrics are like: API run time, API T/L time, API memory utilization, API CPU utilization (rate), API network I/O bytes (and rate).

Next, we have to dockerize and spawn a number of FastAPI servers using different ports.

2

Result and conclusion

Below are some pictures of Grafana UI.

Figure 2.1: Grafana UI for memory utilization, API runtime, etc

Figure 2.2: Grafana UI for CPU utilization, Client IP count etc

Prometheus UI.

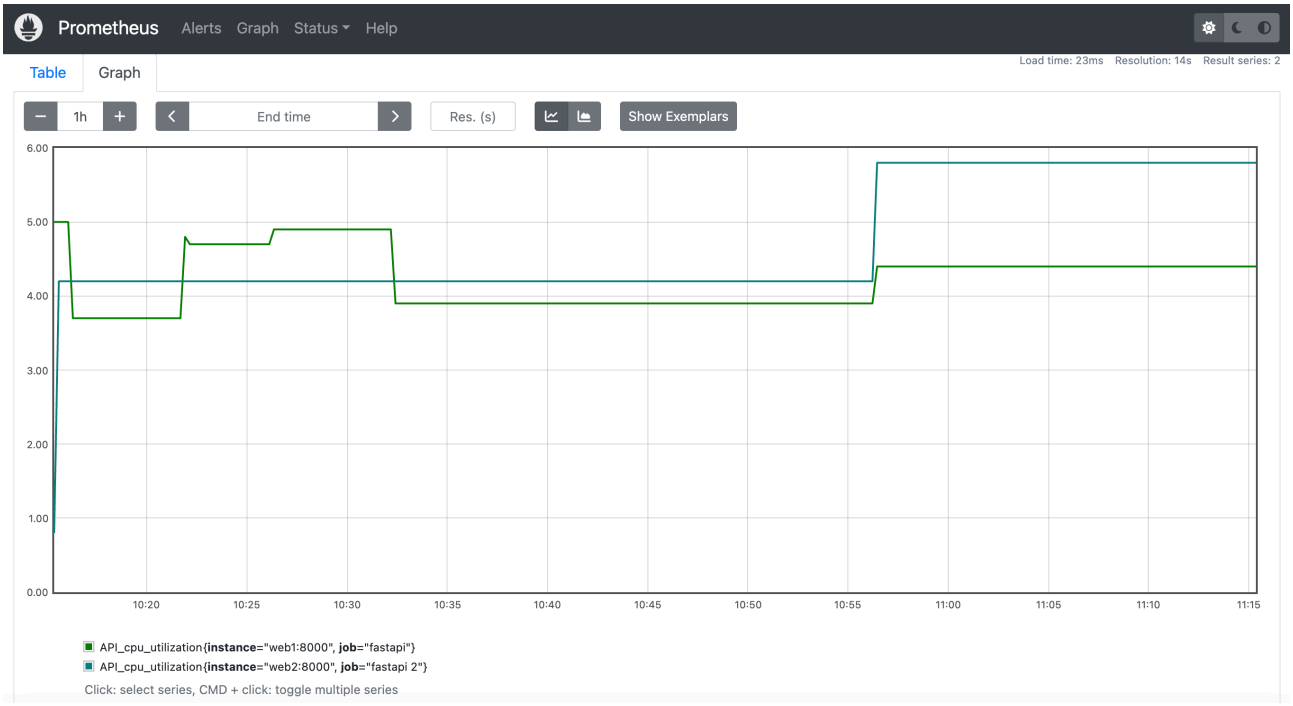


Figure 2.3: Prometheus UI for API CPU utilization

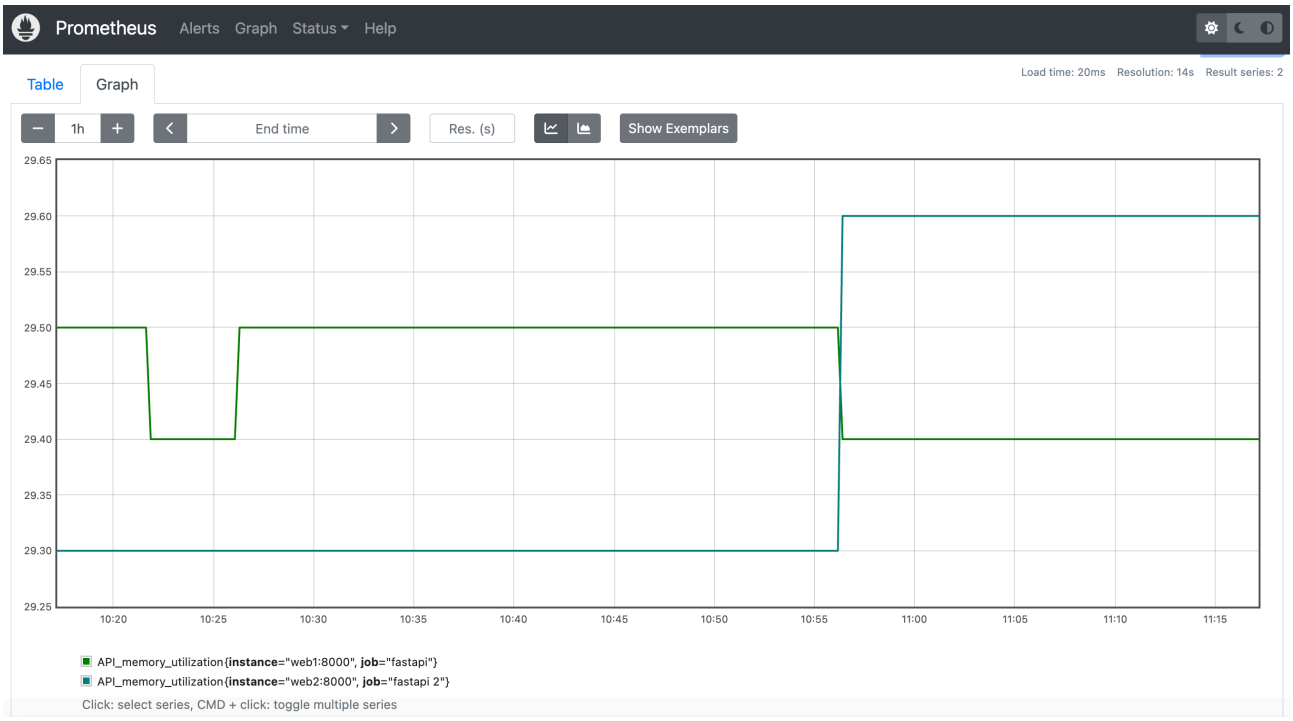


Figure 2.4: Prometheus UI for API memory utilization

For custom handwritten digits:

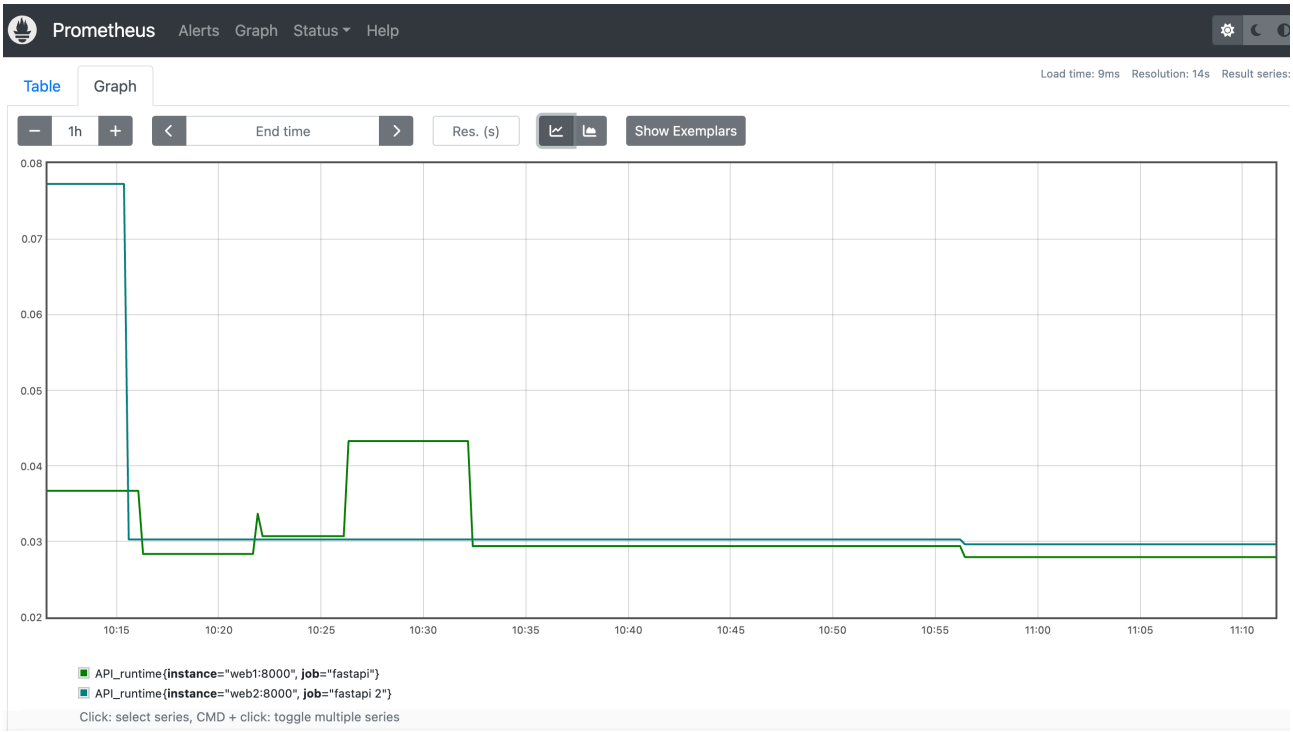


Figure 2.5: Prometheus UI for API runtime

A

Code section

Fast Api code

```
import os
from typing import List
from fastapi import FastAPI, UploadFile, File, HTTPException, Request
import numpy as np
from tensorflow.keras.models import load_model
import tensorflow as tf
from PIL import Image, ImageOps
from io import BytesIO
import sys
import time
import base64
import psutil
from prometheus_fastapi_instrumentator import Instrumentator
from prometheus_client import Counter, Gauge


# Create the FastAPI app
app = FastAPI()

# Load the model from the specified path
def get_model(path: str):
    return load_model(path)

# Load the MNIST model

model_path = "MNIST_model.keras"
#load the pretrained model
model = get_model(model_path)
# set the model in inference mode
model.trainable=False
#counts the number of times a client ip address visit
request_counter = Counter("Client_IP_count", "Total number of requests", ["client_ip"])
#Calculates the run time of API
inference_time_gauge = Gauge("API_runtime", "Inference time in seconds")
#processing time of the text, per character calculation
processing_time_per_char_gauge = Gauge("processing_time_per_char_microseconds", "Processing time per character in microseconds")

# get API network I/O bytes
network_receive_bytes = Gauge("network_receive_bytes", "Total network receive bytes")
network_transmit_bytes = Gauge("network_transmit_bytes", "Total network transmit bytes")

# get the API memory utilization
memory_utilization = Gauge("API_memory_utilization", "API memory utilization")
# get the CPU utilization
cpu_utilization = Gauge("API_cpu_utilization", "API CPU utilization")


# Function to preprocess image and make prediction


def predict_digit(model, data_point):

    # get the prediction containg the score
```

```
    pred = model.predict(data_point)
    # get the class label
    prediction = tf.argmax(pred,axis=-1)
    c_score = np.max(pred)# store the confidence score
    return str(prediction[0].numpy()),str(c_score)

# API endpoint to accept image upload and return prediction
def format_image(image):
    """
    get a pillow image
    """
    # resize the image in 28X28 format
    return image.resize((28,28))

Instrumentator().instrument(app).expose(app)

@app.post('/predict')
async def predict(request: Request, file: UploadFile = File(...)):
    # load the image in the byte format
    content = await file.read()
    accepted_formats = ['.jpeg', '.jpg', '.png']
    file_format = os.path.splitext(file.filename)[1].lower()
    # check for the image file is valid or not
    if file_format not in accepted_formats:
        # raise the error message if the file is wrong in format
        raise HTTPException(status_code=400, detail="Bad␣file␣format.␣Accepted␣formats␣are␣.
            jpeg,␣.jpg,␣.png")
    file_name = os.path.splitext(file.filename)[0]
    image = Image.open(BytesIO(content))
    #convert the image to gray scale first
    image = image.convert('L')
    if image.size!=(28,28):
        # if the image is not 28 by 28 resize it
        image = format_image(image)
    flat = np.array(image,dtype='float32').reshape(-1)/255.0# flatten the image and normalize
         in 0 to 1 scale
    flat = flat[None,:]

    client_ip = request.client.host
    request_counter.labels(client_ip=client_ip).inc()

    start_time = time.time()
    output, c_score = predict_digit(model, flat)
    end_time = time.time()
    inference_time_gauge.set(end_time - start_time)


    # Get memory utilization and update Prometheus metric
    memory_utilization.set(psutil.virtual_memory().percent)

    # Get CPU utilization and update Prometheus metric
    cpu_utilization.set(psutil.cpu_percent())

    input_length = len(file_name)
    processing_time_per_char = (end_time - start_time) / input_length * 1e6  # microseconds
        per character
    processing_time_per_char_gauge.set(processing_time_per_char)

    net_io = psutil.net_io_counters()
    network_receive_bytes.set(net_io.bytes_recv)
    network_transmit_bytes.set(net_io.bytes_sent)
    return {
        "actual":file_name,
        "predicted": output,
        "confidence":c_score}
```