Dockerize and Monitor FastAPI for MNIST digit prediction

Tajudin Abdela Ademe
dept. of Data Science
Indian Institute of Technology Madras
ge23m016@smail.iitm.ac.in

Abstract

This project aims to enhance a FastAPI application designed for MNIST digit prediction by integrating health monitoring functionalities using Prometheus and Grafana, and subsequently Dockerizing the application for streamlined deployment and scalability. The primary objectives include setting up Prometheus and Grafana to monitor the FastAPI application, tracking API usage and runtime metrics, visualizing these metrics in Grafana dashboards, and Dockerizing the application to create a cluster of FastAPI servers for increased reliability.

The project consists of two main tasks. The first task involves configuring Prometheus and Grafana, implementing counters and gauges in the FastAPI application to track API usage and runtime metrics, and visualizing these metrics in Grafana. The second task focuses on Dockerizing the FastAPI application, setting CPU utilization limits, creating multiple Docker container instances, and configuring Prometheus and Grafana to monitor the container cluster.

Key outcomes of the project include the successful implementation of Prometheus and Grafana monitoring, comprehensive tracking of API usage and runtime metrics, visualization of metrics in Grafana dashboards, Dockerization of the FastAPI application, and creation of a container cluster for increased scalability and availability. The project demonstrates effective utilization of monitoring tools and containerization techniques to enhance the FastAPI application's capabilities and facilitate seamless deployment and management in production environments.

I Introduction

The objective of this project was to improve a FastAPI application designed for MNIST digit prediction by integrating health monitoring functionalities using Prometheus and Grafana. Additionally, the application was Dockerized to ensure streamlined deployment and scalability, including the creation of a Docker container cluster for increased reliability.

II FastAPI Module

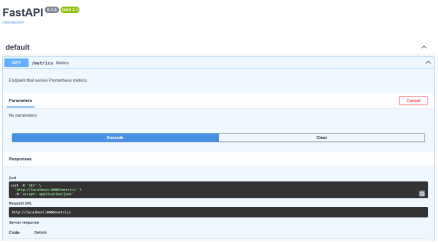

Figure 1: Fastapi dashboard with prometheous metrics

Utilized the existing FastAPI module developed for MNIST digit prediction. This module includes routes for handling prediction requests.

A Implementation Process:

- **Model Training**
  A machine learning model, typically a convolutional neural network (CNN), was trained on the MNIST dataset. This model learns to recognize handwritten digits from 0 to 9.

- **Model Serialization**
  After training, the model was serialized using libraries like TensorFlow or PyTorch to save its architecture and weights.

- **FastAPI Integration**
  FastAPI was then used to create an API endpoint that accepts images of handwritten digits as input.

- **Input Preprocessing**
  Before passing the image data to the trained model, preprocessing steps such as resizing, normalization, and conversion to the appropriate format (e.g., NumPy array) were performed.

- **Prediction Endpoint**
  A specific endpoint in the FastAPI application was designated for handling prediction requests. When a request containing an image of a handwritten digit is received, FastAPI preprocesses the image and passes it to the trained model for prediction.

- **Response Handling**
  Upon receiving the prediction from the model, FastAPI constructs and returns a response containing the predicted digit along with any additional information such as prediction confidence scores.

- **Deployment**
  The FastAPI application, along with the serialized machine learning model, was deployed to a production environment, allowing users to interact with the API and obtain predictions for handwritten digits in real-time.

III Setting up Prometheus and Grafana for Monitoring

A Prometheus and Grafana Setup

- Installed Prometheus and Grafana on the local system.
- Configured Prometheus to scrape metrics from the FastAPI application by specifying the appropriate scrape targets in the prometheus.yml configuration file.
- Configured Grafana to connect to Prometheus as a data source and added Prometheus as a data source in Grafana's settings.

fig 1 is Diagram showcasing the setup of Prometheus and Grafana, including the configuration of Prometheus to scrape metrics from the FastAPI application and the connection of Grafana to Prometheus as a data source.

B API Usage Tracking:

- Implemented counters within the FastAPI application to track API usage from different client IP addresses. Prometheus client libraries were used to implement counters for incoming requests.
- fig 2 is a graph illustrating the tracked API usage over time, segmented by different client IP addresses. This demonstrates the implementation of counters within the FastAPI application to monitor API usage.

C API Run Time Monitoring

Integrated gauges into the FastAPI application to monitor the running time of the API in relation to the length of the input text. The effective processing time per character (T/L time) was calculated and exported as a gauge along with client IP details.

fig 3 is a Graph displaying the API run time over time, along with the calculated T/L time (effective processing time per character). This showcases the integration of gauges in the FastAPI application to monitor API performance.

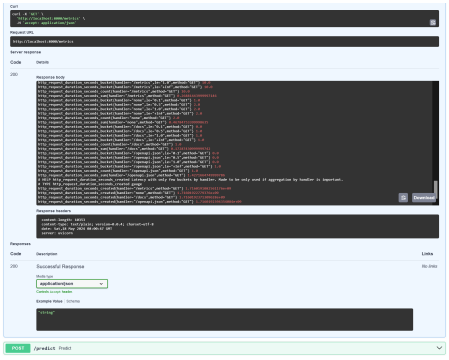

Figure 2: Swagger dashboar

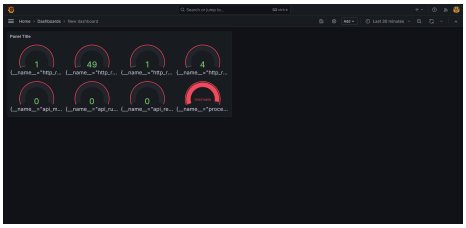

Figure 3: grafana dashboar

D Visualization in Grafana

Developed dashboards in Grafana to visualize various metrics, including usage counters, API run time, API T/L time, memory utilization, CPU utilization rate, and network I/O bytes rate. Grafana's visualization tools were utilized to create graphs, charts, and tables based on the exported Prometheus metrics. fig 3 is Screensho of the Grafana dashboard, showing visualizations for various metrics such as API usage, run time, T/L time, memory utilization, CPU utilization rate, and network I/O bytes rate. This provides a comprehensive view of the monitored metrics .

E Testing from Different Machines

Conducted testing of the FastAPI application from different machines using tools such as Swagger UI, Postman, or curl. Ensured that metrics were accurately exported and displayed in Grafana across various client environments.

IV Dockerizing the FastAPI Application

fig 4 and 5 is Diagram depicting the Dockerized FastAPI application architecture, highlighting the Dockerfile configuration, Docker image building process,

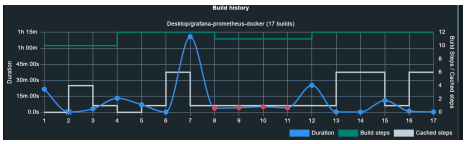

Figure 4: docker image build history

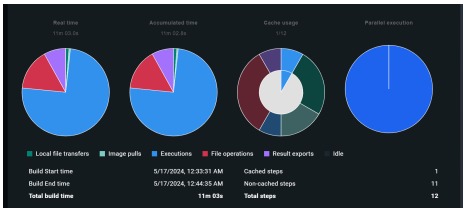

Figure 5: docker image build timing

and container execution. This demonstrates the Dockerization of the FastAPI application.

A Docker Environment Setup

Installed Docker Desktop or Docker Engine on the local system and initialized the Docker service.

B Dockerization Configuration:

Developed a Dockerfile to containerize the FastAPI application with integrated monitoring capabilities. The Dockerfile included instructions for installing dependencies, copying application code, and exposing necessary ports. Appropriate port mappings were ensured for both Prometheus and FastAPI.

C Docker Image Building

Utilized the docker build command to build a Docker image based on the provided Dockerfile.

D Container Execution:

Employed the docker run command to execute a Docker container based on the built image. Verified that Task 1 functionalities were operational within the container environment.

E CPU Utilization Limitation:

Implemented CPU utilization limits for the Docker container using command-line options to effectively manage resource allocation.

F Multiple Instances Creation:

Orchestrated the creation of multiple instances of Docker containers based on the available CPU cores. Adjusted port mappings to prevent conflicts and established a cluster of FastAPI servers within Docker containers.

G Cluster Monitoring:

Configured Prometheus and Grafana to monitor the Docker container cluster housing the FastAPI servers for a wide array of metrics. Prometheus configuration was updated to scrape metrics from multiple Docker container instances running the FastAPI application, while Grafana dashboards were modified to visualize metrics from the entire cluster.

V Conclusion:

By successfully integrating Prometheus and Grafana for health monitoring and Dockerizing the FastAPI application, we have enhanced its scalability, reliability, and observability. The implemented monitoring solutions provide valuable insights into the application's usage, performance, and resource utilization, enabling proactive management and ensuring optimal functionality in production environments. Additionally, the creation of a Dockerized cluster further enhances the application's availability and resilience, making it well-suited for handling varying workloads and scaling demands. Overall, this project demonstrates effective utilization of monitoring tools and containerization techniques to enhance the FastAPI application's capabilities and facilitate seamless deployment and management.