Department Of Aerospace Engineering
Indian Institute Of Technology Madras

CS5830: Big Data Laboratory
Prof. Sudarsun

Lab Assignment 7:
FastAPI Monitoring with Prometheus and Grafana

Sanket Pramod Bhure (*AE20B108*)

May 17, 2024

Contents

1 Introduction

This project aims to enhance a FastAPI application by integrating Prometheus for API monitoring and Grafana for visualization. The project also includes dockerization and scaling to multiple instances for comprehensive cluster health monitoring. Grafana is utilized for visualizing the Prometheus metrics. The assignment is available in the repository Sanky18/CS5830-Big-Data-Laboratory-Assignment-7 on GitHub.

2 Features

- **Prometheus Monitoring**: Track API usage and monitor health metrics.
- **Grafana Visualization**: Visualize Prometheus metrics for insights into API performance.
- **Dockerization**: Seamlessly build and run the application in Docker containers.
- **Scaling**: Deploy multiple instances of the FastAPI application to form a cluster.
- **Metrics**: Monitor API usage counters, runtime, memory utilization, CPU utilization, and network I/O.

3 Project Structure

```
Experiments
Result
   dashboard_image_1.png
   dashboard_image_2.png
   ...
app
   Dockerfile
   digit_predictor.py
   docker-compose.yml
   image_formatter.py
   main.py
   mnist_model.h5
   model_loader.py
   prometheus.yml
   requirements.txt
   testing.py
models
scripts
   train_best_model.py
tests
```

4 Procedure

To set up the project and run it on your local machine, follow these steps:

- **Clone the Repository:**
  First, clone the repository to your local machine using the following command:

```
git clone
https://github.com/Sanky18/CS5830-Big-Data-Laboratory-Assignment-7.git
```

- **Install Dependencies:**
  Navigate to the `app` directory of the cloned repository and install the required dependencies by running the following command:

  ```
  cd app
  pip install -r requirements.txt
  ```

  This will ensure all necessary Python packages are installed.

- **Prometheus Configuration:**
  Ensure that the `prometheus.yml` file is located in the `app` directory. This file contains the configuration for Prometheus monitoring.

- **Docker Setup:**
  Create a `Dockerfile` in the `app` directory to define the dockerization configuration. This file specifies how to build the Docker image for the FastAPI application.

- **Docker Compose Configuration:**
  Create a `docker-compose.yml` file in the `app` directory to define the services and ports for Docker Compose. This file specifies how to run multiple Docker containers and link them together.

- **Build and Run the Docker Containers:**
  Finally, build and run the Docker containers using the following command:

  ```
  docker-compose up --build
  ```

  This command will build the Docker image and start the containers based on the configurations specified in the `docker-compose.yml` file.

After running these steps, we should be able to access the FastAPI application, Prometheus, and Grafana from respective local host websites.

5 Result

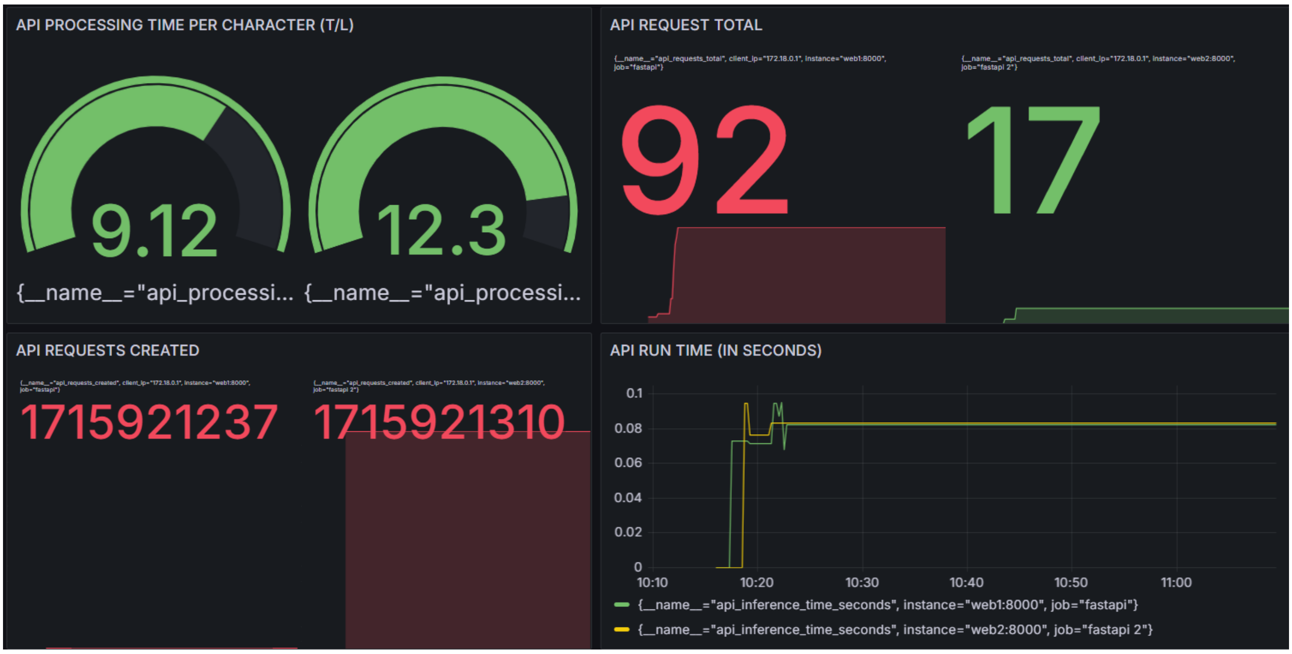
API PROCESSING TIME PER CHARACTER (T/L)
9.12
12.3
{__name__="api_processi...
{__name__="api_processi...
API REQUEST TOTAL
{__name__="api_requests_total", client_ip="172.18.0.1", instance="web1:8000", job="fastapi"}
92
{__name__="api_requests_total", client_ip="172.18.0.1", instance="web2:8000", job="fastapi 2"}
17
API REQUESTS CREATED
{__name__="api_requests_created", client_ip="172.18.0.1", instance="web1:8000", job="fastapi"}
1715921237
{__name__="api_requests_created", client_ip="172.18.0.1", instance="web2:8000", job="fastapi 2"}
1715921310
API RUN TIME (IN SECONDS)
0.1
0.08
0.06
0.04
0.02
0
10:10
10:20
10:30
10:40
10:50
11:00
{__name__="api_inference_time_seconds", instance="web1:8000", job="fastapi"}
{__name__="api_inference_time_seconds", instance="web2:8000", job="fastapi 2"}

Figure 1: Grafana dashboard displaying various metrics: Processing Time per Character, API Requests Created, API Request Total, and API Run Time. These visualizations provide insights into the performance and usage of the FastAPI application.

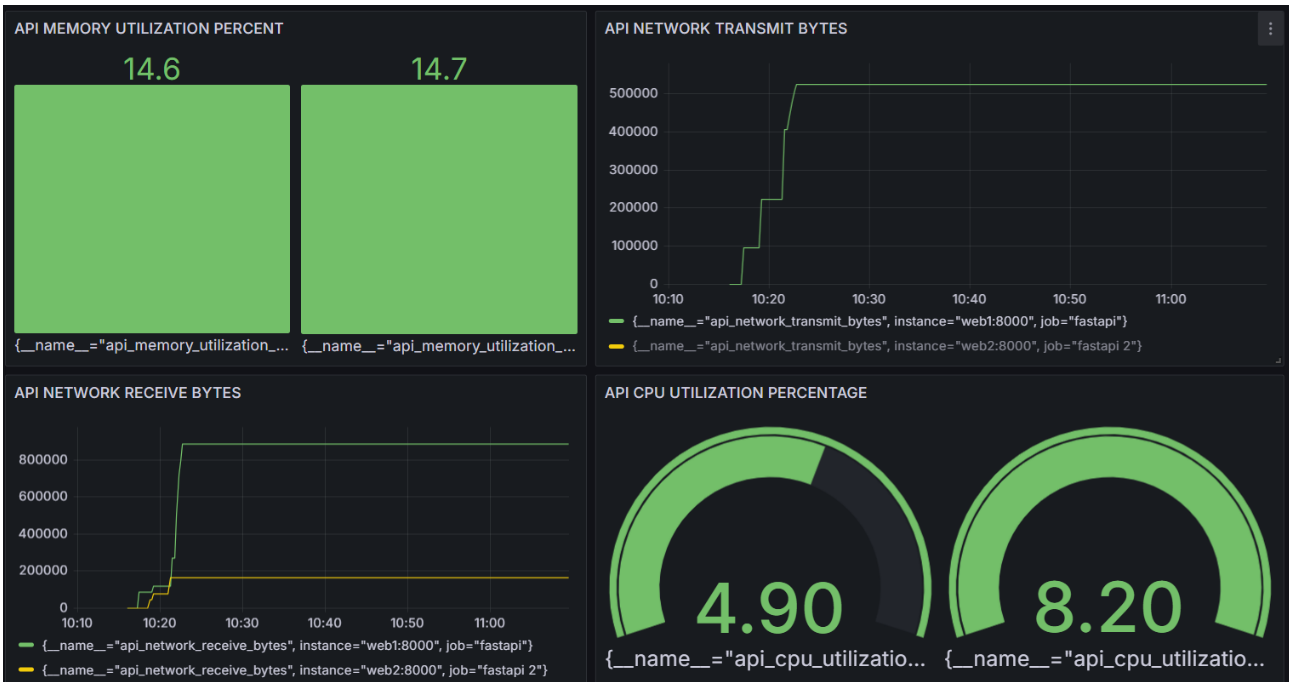
API MEMORY UTILIZATION PERCENT
14.6
14.7
{__name__="api_memory_utilization_...
{__name__="api_memory_utilization_...
API NETWORK TRANSMIT BYTES
500000
400000
300000
200000
100000
0
10:10
10:20
10:30
10:40
10:50
11:00
{__name__="api_network_transmit_bytes", instance="web1:8000", job="fastapi"}
{__name__="api_network_transmit_bytes", instance="web2:8000", job="fastapi 2"}
API NETWORK RECEIVE BYTES
800000
600000
400000
200000
0
10:10
10:20
10:30
10:40
10:50
11:00
{__name__="api_network_receive_bytes", instance="web1:8000", job="fastapi"}
{__name__="api_network_receive_bytes", instance="web2:8000", job="fastapi 2"}
API CPU UTILIZATION PERCENTAGE
4.90
8.20
{__name__="api_cpu_utilizatio...
{__name__="api_cpu_utilizatio...

Figure 2: Grafana dashboard displaying various metrics: API Memory Utilization Percent, API Network Transmit Bytes, API Network Receive Bytes, and API CPU Utilization Percentage. These visualizations provide insights into the performance and usage of the FastAPI application.

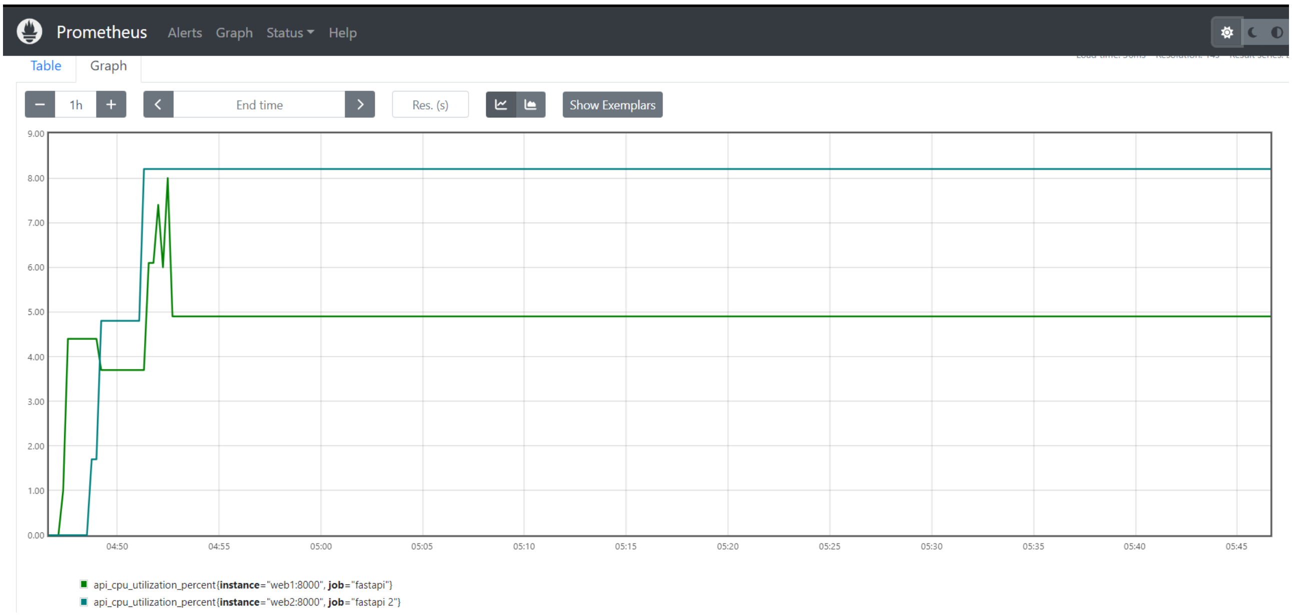

Figure 3: Prometheus plot displaying the `api_cpu_utilization` metric. This metric monitors the CPU usage of the FastAPI application, providing insights into its computational load and performance.

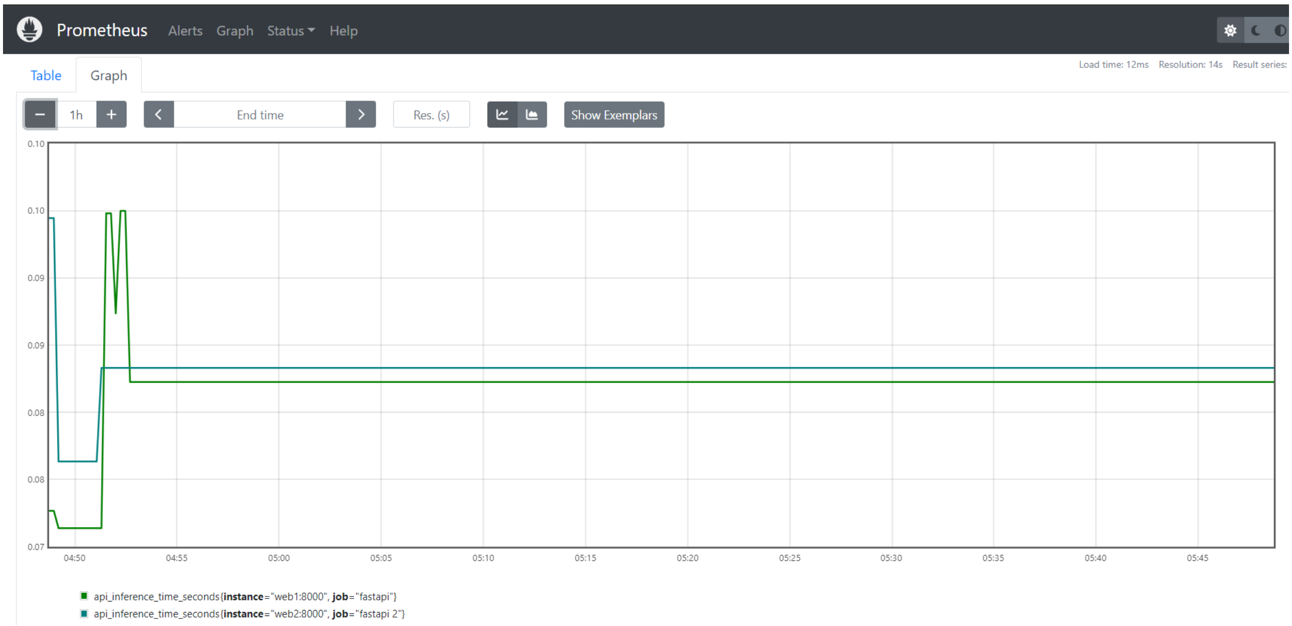

Figure 4: Prometheus plot displaying the `api_inference_time` metric. This metric monitors the CPU usage of the FastAPI application, providing insights into its computational load and performance.

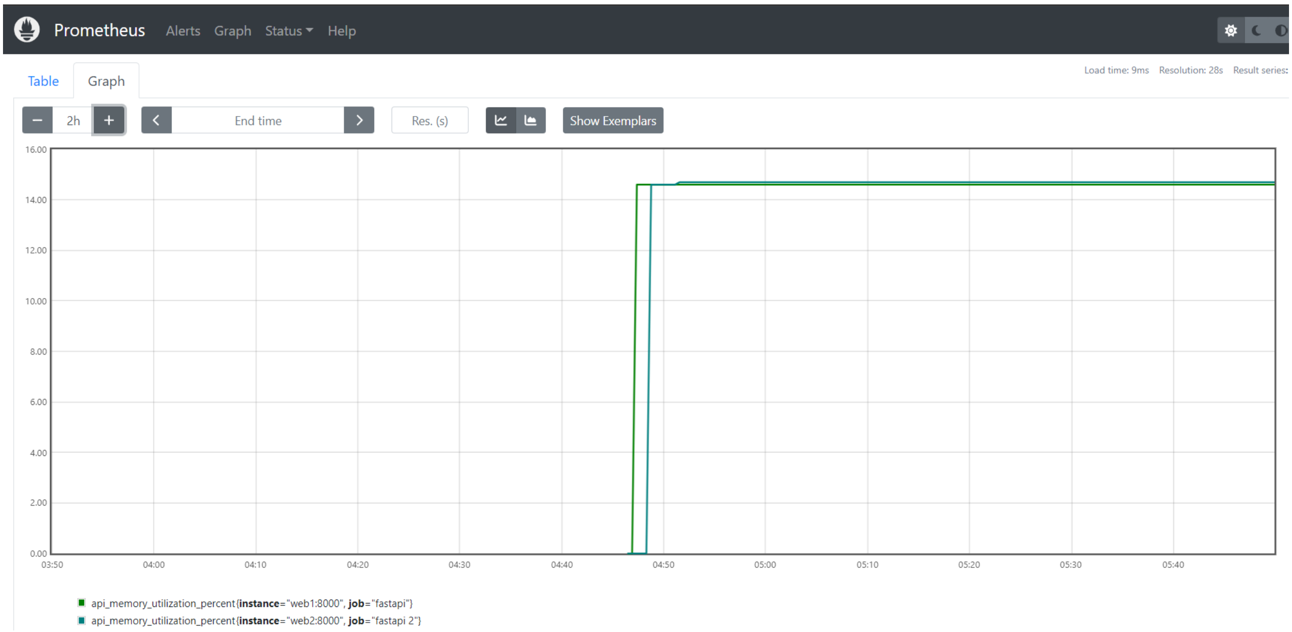

Figure 5: Prometheus plot displaying the `api_memory_utilization` metric. This metric monitors the CPU usage of the FastAPI application, providing insights into its computational load and performance.

6 Conclusion

This project demonstrates how to enhance a FastAPI application with monitoring and visualization capabilities using Prometheus and Grafana. By dockerizing the application and scaling it to multiple instances, we can effectively monitor and manage a clustered environment's health.