Indian Institute of Technology Jodhpur

Fundamentals of Distributed Systems

Assignment– 1

Student Name: Avinash Kumar
Roll Number: G24AI2025.

Q.2 Dynamic Load Balancing for a Smart Grid.

Introduction:

In a future powered by electric vehicles (EVs), the demand on charging infrastructure will grow exponentially. Imagine dozens—or hundreds—of EVs trying to charge simultaneously across a city. If all requests are routed without intelligence, some substations may become overwhelmed while others remain underused. This can lead to grid instability, longer wait times, and inefficient resource utilization.

This project tackles that very challenge by simulating a smart, load-aware EV charging system. It acts like a traffic controller for electricity—monitoring real-time load across substations and ensuring that each new request is routed to the least loaded substation.

What makes this system intelligent is:

- Its use of live monitoring (via Prometheus),

- A custom-built load balancer that makes real-time routing decisions,
- And a visual dashboard (Grafana) to observe how load changes as requests flow in.

The entire architecture is written in Python, containerized using Docker, and orchestrated with Docker Compose to mirror a true distributed, scalable system.

2.Objective:

Build a system that can intelligently balance EV charging requests in real time, so no single substation gets overloaded.

But this wasn't just about writing some routing logic. I wanted to simulate a real-world grid-like environment where:

- Charging stations (substations) experience real-time usage variations.
- A central decision-making unit (load balancer) polls and reacts dynamically.
- The system is observable, meaning we can see exactly how traffic is flowing, how load is shifting, and whether the logic is working under pressure.

The vision was to create an end-to-end microservice architecture where:

- Each service had a clear, independent role,
- The system could handle rush hours,
- And the entire infrastructure was modular and scalable through Docker.

3. System Architecture

The architecture is made up of six distinct services working together:

Component	Purpose
charge_request_service	Entry point for all charging requests
load_balancer	Acts as the brain: polls loads and routes requests intelligently
substation_service	Simulates real-world charging and exposes real-time load metrics
load_tester	Generates a flood of charging requests to simulate traffic surges
prometheus	Continuously scrapes metrics from substations for monitoring
grafana	Provides real-time visualization of how the system is performing

Each substation works as an independent microservice. When a request arrives:

1. It goes to the **charge_request_service**.
2. The request is then forwarded to the **load balancer**.
3. The balancer consults all **substation metrics** and routes the request to the **least busy** one.

4. Prometheus scrapes the latest loads.
5. Grafana visualizes the whole system in real time.

4. Screenshots.

The below screenshot shows

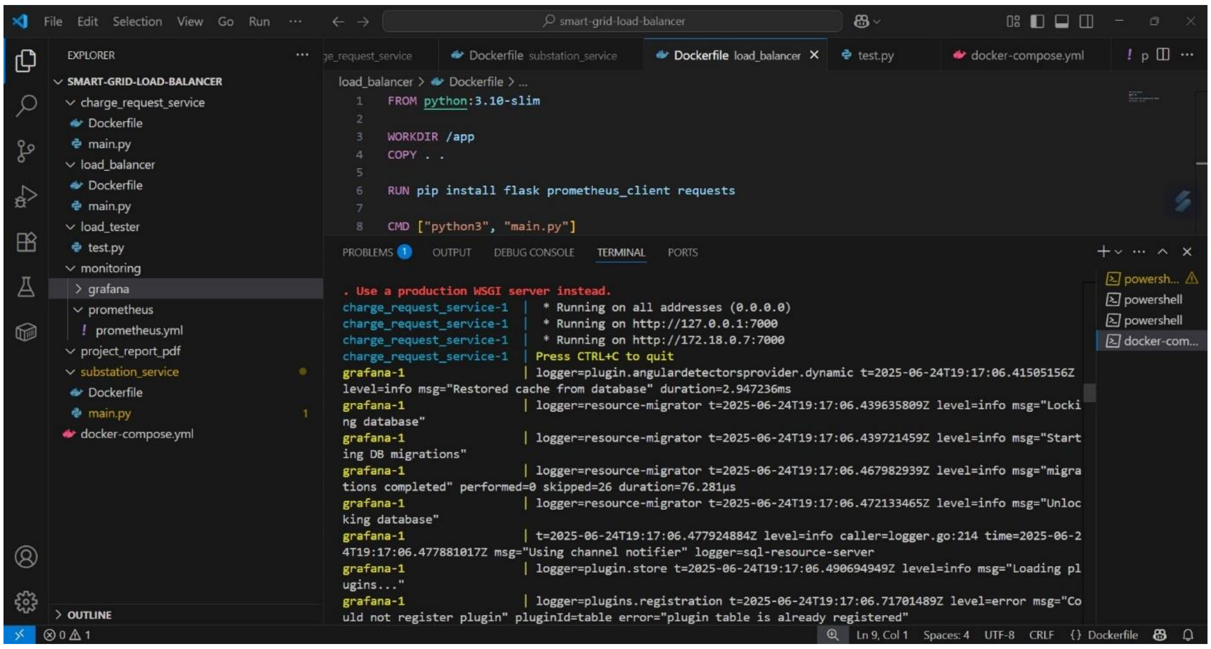
The left pane reveals a well-organized folder structure with clearly separated services: charge_request_service, load_balancer, substation_service, and monitoring.

The Dockerfile for load_balancer is visible, specifying Python 3.10 and dependencies like flask, prometheus_client, and requests.

The terminal confirms that:

- charge_request_service is live and listening on port 7000 (internally and externally).
- grafana-1 is booting successfully and migrating the internal database.

Prominent log messages show that Grafana is restoring from cache, migrating plugins, and initializing correctly.



```
FROM python:3.10-slim

WORKDIR /app
COPY . .

RUN pip install flask prometheus_client requests

CMD ["python3", "main.py"]
```

```
. Use a production WSGI server instead.
charge_request_service-1  |  * Running on all addresses (0.0.0.0)
charge_request_service-1  |  * Running on http://127.0.0.1:7000
charge_request_service-1  |  * Running on http://172.18.0.7:7000
charge_request_service-1  | Press CTRL+C to quit
grafana-1                 | logger=plugin.angulardetectorsprovider.dynamic t=2025-06-24T19:17:06.41505156Z
level=info msg="Restored cache from database" duration=2.947236ms
grafana-1                 | logger=resource-migrator t=2025-06-24T19:17:06.439635809Z level=info msg="Locki
ng database"
grafana-1                 | logger=resource-migrator t=2025-06-24T19:17:06.439721459Z level=info msg="Start
ing DB migrations"
grafana-1                 | logger=resource-migrator t=2025-06-24T19:17:06.467982939Z level=info msg="migra
tions completed" performed=0 skipped=26 duration=76.281µs
grafana-1                 | logger=resource-migrator t=2025-06-24T19:17:06.472133465Z level=info msg="Unloc
king database"
grafana-1                 | t=2025-06-24T19:17:06.477924884Z level=info caller=logger.go:214 time=2025-06-2
4T19:17:06.477881017Z msg="Using channel notifier" logger=sql-resource-server
grafana-1                 | logger=plugin.store t=2025-06-24T19:17:06.490694949Z level=info msg="Loading pl
ugins..."
grafana-1                 | logger=plugins.registration t=2025-06-24T19:17:06.71701489Z level=error msg="Co
uld not register plugin" pluginId=table error="plugin table is already registered"
```

The below screenshot says that

The terminal logs are from substation1-1.

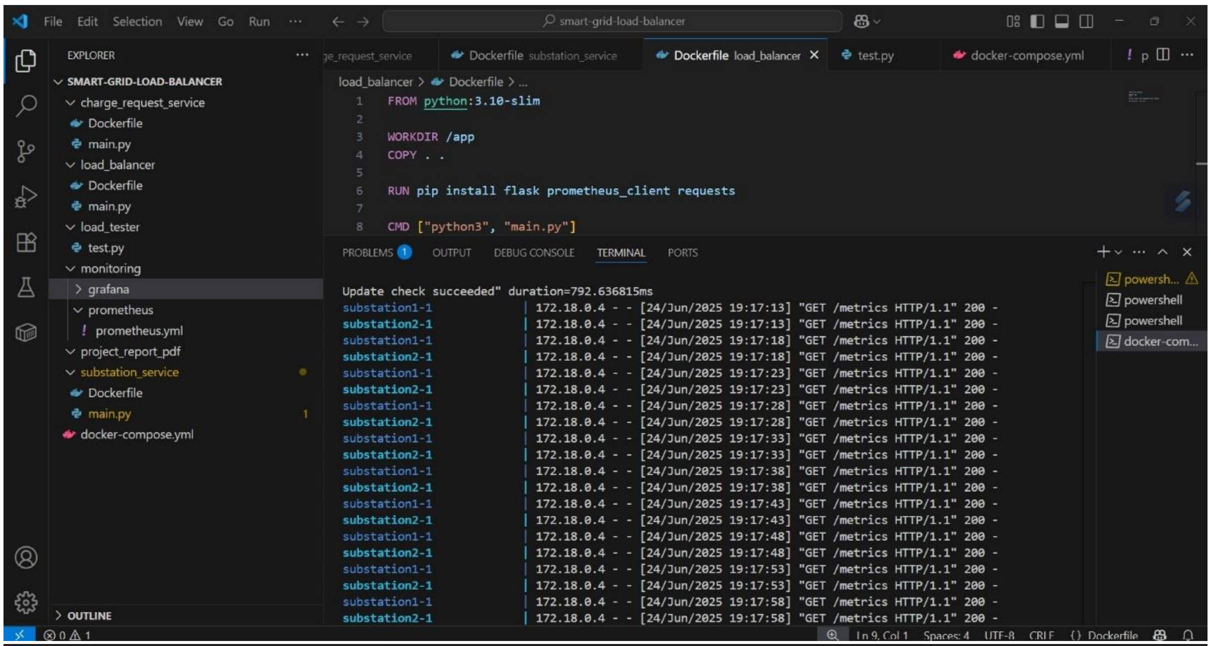
Prometheus is sending repeated GET /metrics requests.

The HTTP responses are all 200 OK, indicating successful retrieval.

Its also validating

- The observability pipeline is functional.
- Prometheus is collecting real-time load data from each substation.
- The load balancer and Grafana can now use this data to:
  - Make routing decisions
  - Display live dashboards

Without this metric flow, the system would be blind to substation states. This screenshot proves full integration between application logic and monitoring stack.



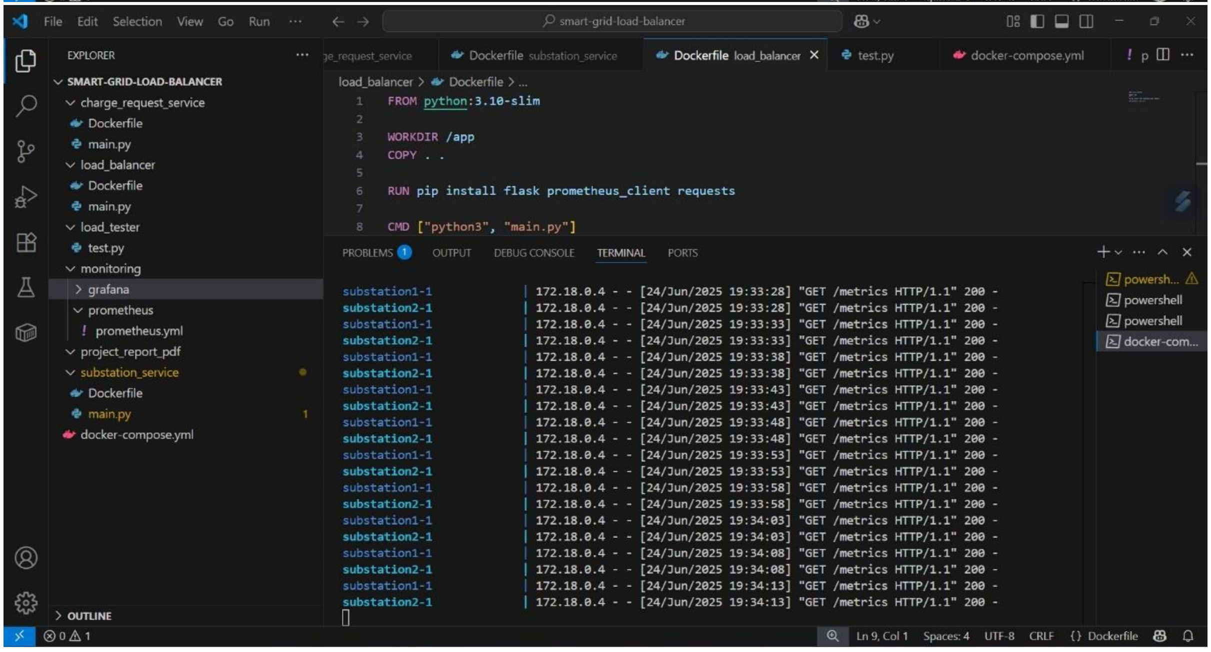
```
FROM python:3.10-slim

WORKDIR /app
COPY . .

RUN pip install flask prometheus_client requests

CMD ["python3", "main.py"]
```

```
Update check succeeded" duration=792.636815ms
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:17:13] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:17:13] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:17:18] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:17:18] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:17:23] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:17:23] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:17:28] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:17:28] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:17:33] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:17:33] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:17:38] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:17:38] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:17:43] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:17:43] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:17:48] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:17:48] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:17:53] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:17:53] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:17:58] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:17:58] "GET /metrics HTTP/1.1" 200 -
```



```
FROM python:3.10-slim

WORKDIR /app
COPY . .

RUN pip install flask prometheus_client requests

CMD ["python3", "main.py"]
```

```
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:33:28] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:33:28] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:33:33] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:33:33] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:33:38] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:33:38] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:33:43] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:33:43] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:33:48] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:33:48] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:33:53] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:33:53] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:33:58] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:33:58] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:34:03] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:34:03] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:34:08] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:34:08] "GET /metrics HTTP/1.1" 200 -
substation1-1          | 172.18.0.4 - - [24/Jun/2025 19:34:13] "GET /metrics HTTP/1.1" 200 -
substation2-1          | 172.18.0.4 - - [24/Jun/2025 19:34:13] "GET /metrics HTTP/1.1" 200 -
```

The below screenshot says

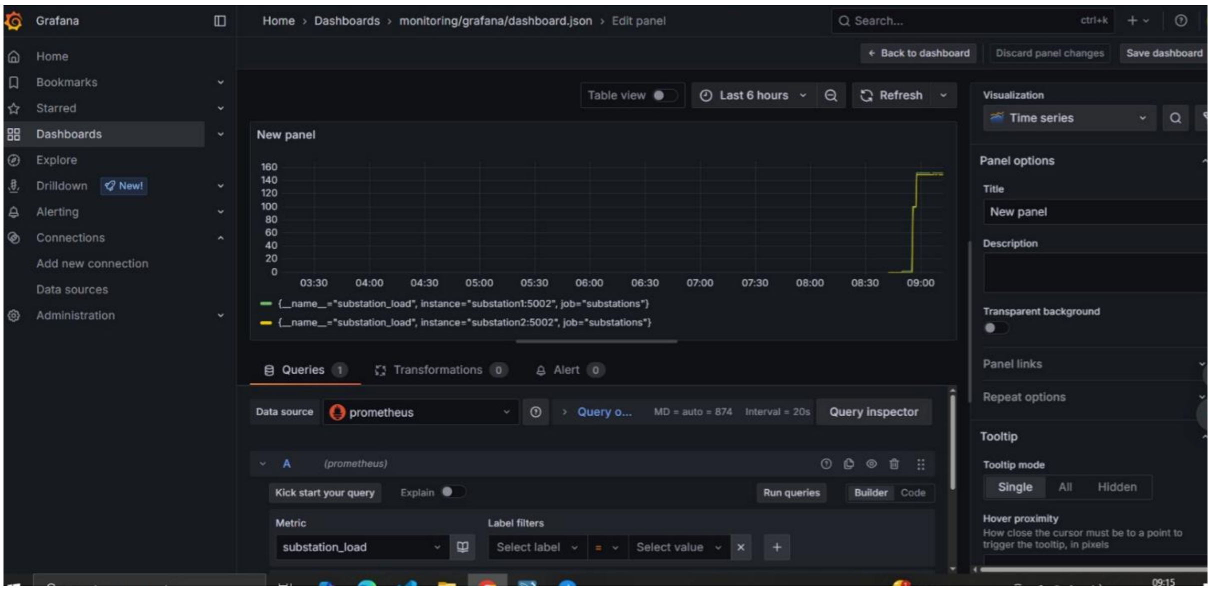
The query substation_load (bottom section) is fetching live metrics from Prometheus.

The line graph shows a **step-wise increase in load**, as charging requests are distributed across substations (substation1 and substation2).

Two colored lines:

- **Yellow**: Substation 1
- **Green**: Substation 2

You can clearly observe when a **rush of requests** was received, leading to sharp upward curves.



New panel

{__name__="substation_load", instance="substation1:5002", job="substations"}
{__name__="substation_load", instance="substation2:5002", job="substations"}

Data source: prometheus
Query o... MD = auto = 874 Interval = 20s
A (prometheus)
Metric: substation_load
Label filters: Select label = Select value

Visualization: Time series
Panel options
Title: New panel
Description:
Transparent background
Tooltip mode: Single
Hover proximity: How close the cursor must be to a point to trigger the tooltip, in pixels

Video Link :

https://drive.google.com/file/d/14-8PPYa8LtkCgItV0L9jo-QbGuwOAQqV/view?usp=sharing