Indian Institute of Technology Jodhpur

Fundamentals of Distributed Systems

Assignment – 1, Task - 2

Name: Ashutosh Tripathi

Roll no.: G24AI2076

Dynamic Load Balancing for a Smart Grid

1. Introduction

In this project, a system that could handle incoming electric vehicle (EV) charging requests and distribute them efficiently across multiple substations has been created. Instead of letting one substation handle too much load while others sit idle, the project uses live metrics to check how busy each station is and makes a smart decision about where to send the next EV. We used Docker to containerize each part of the system and added monitoring through Prometheus and Grafana to observe how the load was being handled in real-time.

2. System Architecture

The architecture follows a microservices model where different services run independently and communicate with each other.
Each of the microservice runs as a Docker Container and is defined in docker-compose.yml. The services communicate to each other over a virtual Docker network (hostname), and each has a specific role:

2.1. Charge Request Service

Logic File: charge_request_service/main.py

This is the entry point of the system. Whenever an EV wants to charge, a request is sent here (in our case via a simulated Python script). It doesn't do much processing but simply forwards the request to the load balancer.
Essentially it:

- Accepts POST requests like {"ev_id": "EV1234"}
- Forwards this request to the Load Balancer's /route endpoint

2.2. Load Balancer

Logic File: load_balancer/main.py

This is essentially the most important part of the system. It keeps track of how busy each substation is by checking /metrics collected by Prometheus.
Based on this data, it decides which substation is the least loaded and forwards the EV's request there.

- Every 5 seconds, sends HTTP GET requests to substations /metrics endpoints.

- Extracts the value of the 'substation_current_load' metric from the Prometheur-style response (eg. Substation_current_load 3).
- When a POST request hits /route, it:
  1. Finds the substation with the lowest load.
  2. Forwards EV request to that substation's /charge endpoint.

2.3. Substation Services

Logic File: substation_service/main.py (used for all 3 substations)

Each substation runs the same code base but as separate container with a different name(substation1, substation2, substation3)

- They expose two endpoints:
  - /charge: Accepts a POST request with vehicle ID (Simulates a charging session) by sleeping for x seconds.
  - /metrics: Returns for eg. Substation_current_load 3.
  - Internally maintains a count of how many EVs are currently charging, using a thread safe variable.
- When a charging session starts it:
  - Increments the load.
  - Waits for X seconds (simulates charging)
  - Decrements the load.

2.4. Monitoring Stack

Two open-source tools used to monitor and visualize the system, and verify how balanced the system was during testing:

Prometheus:

- Scrapes metrics from /metrics endpoint of each substation (every 5 seconds)
- Stores historical data like:
  - How did the load vary over time.
  - Peak of the load.

Grafana:

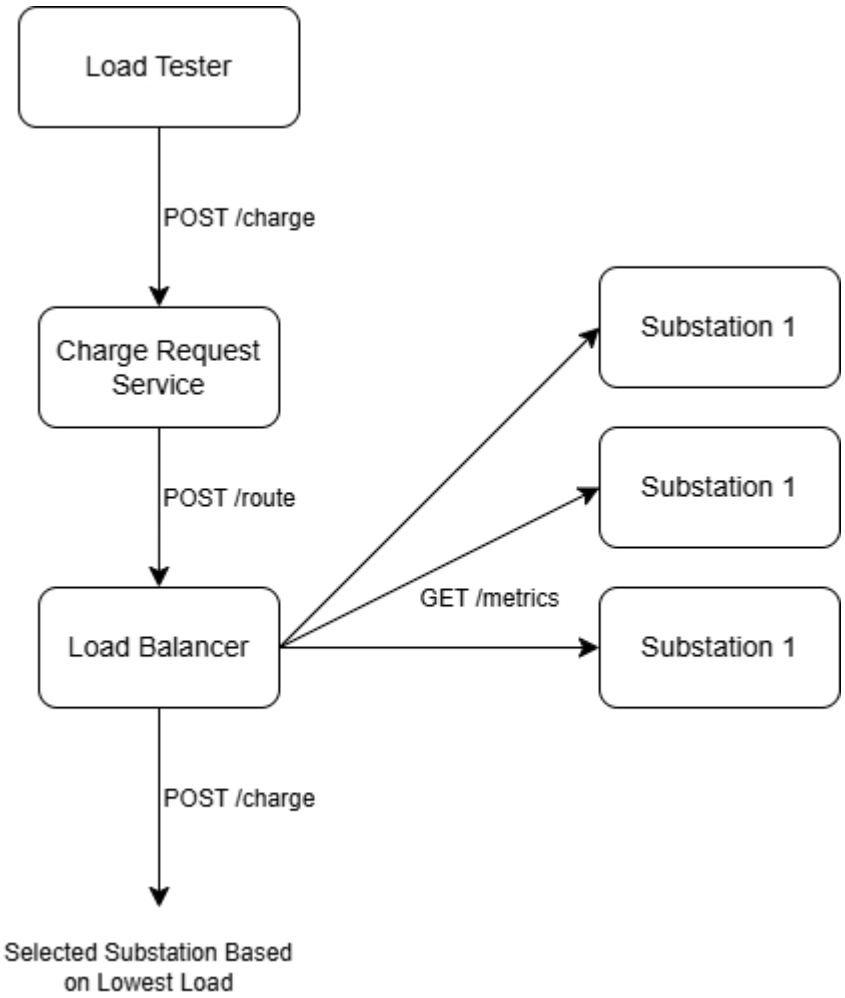
- Displays those collected metrics in a visual format like graphs and gauges.
- Uses Prometheus as its data source.
- Shows:
  - Substation 1/2/3 current load over time(graph)
  - Instant Values (guage).

2.5. Load Tester

Logic File: load_tester/test.py

- Sends a bunch of requests (10-50 EVs) quickly to the Charge Request Service with a generated ev_id.
- Acts as a stress test for the system

ArchitectureDiagram:



3. System Performance Analysis

3.1. Testing Setup

- We simulated a high-load scenario by sending 1 EV request every 2 seconds, for 5 minutes.(load_tester/test.py).
- The system had 3 charging substations, each with a simulated 30-second session duration.

3.2. Expected Behaviour

Based on the system architecture and the way the load_balancer service was designed, the following behaviour is expected:

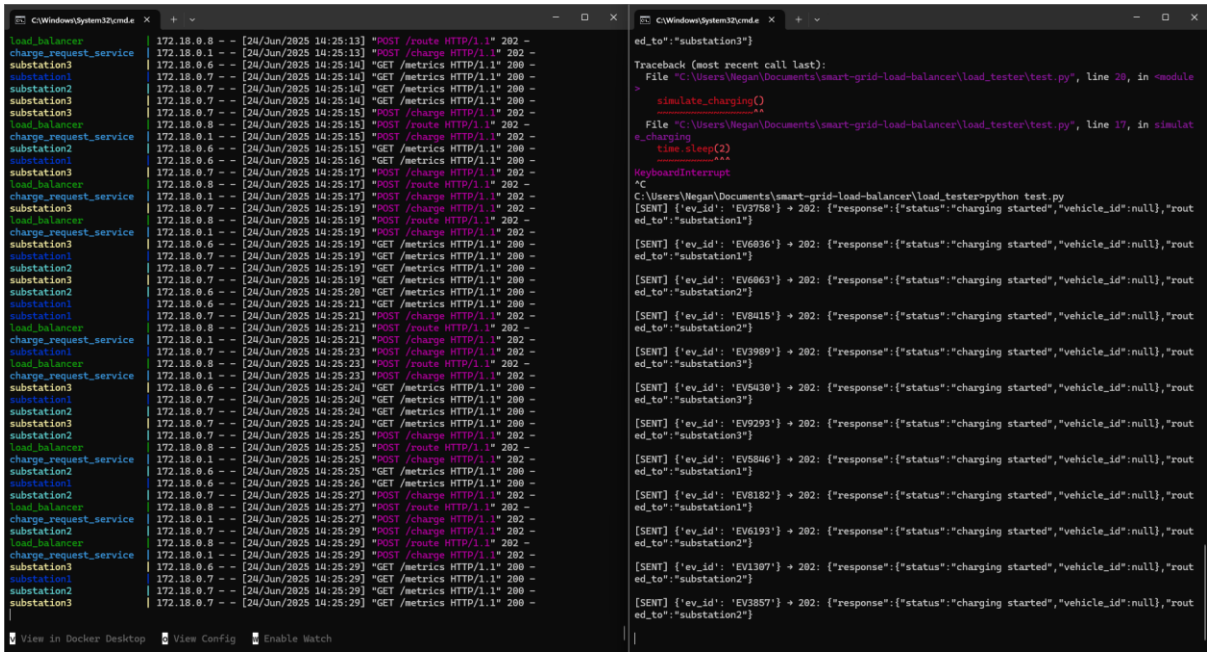
- Each incoming EV request should be routed to the substation currently handling the least number of vehicles.
- The load balancer polls the /metrics endpoint from all three substations every 5 seconds and stores their reported load values.
- Routing decisions are made by comparing those values and choosing the substation with the minimum load.
- Prometheus scrapes the same /metrics data at a regular interval, and Grafana visualizes this data using live dashboards.

3.3. Observed Outcomes

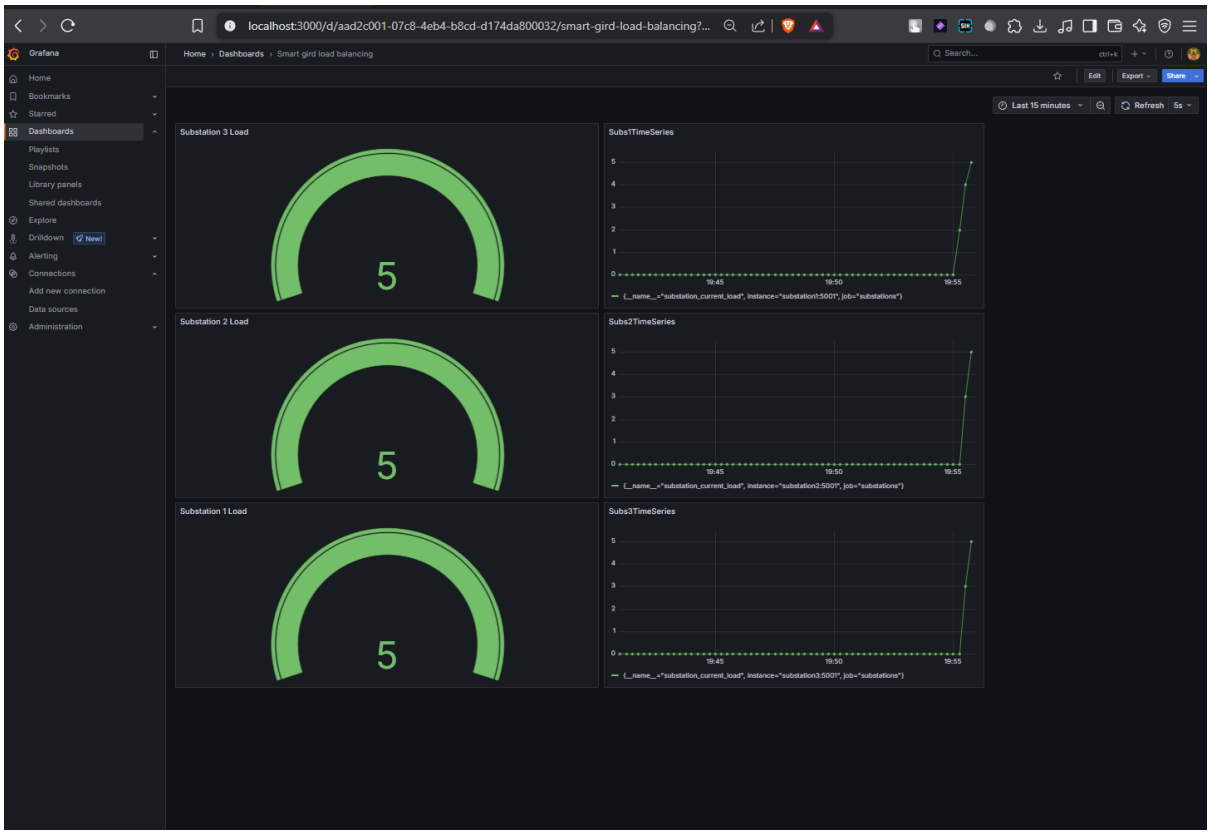
- Upon executing the load testing script, electric vehicle (EV) charge requests were continuously sent to the /charge endpoint. These requests were **successfully routed by the load balancer to the least-loaded substation at the time of each request**.
- vehicles were evenly distributed across all three substations (substation1, substation2, and substation3). Each substation received multiple requests, and **the routing was dynamically adjusted based on current load metrics.**
- As the simulation progressed, **small spikes in load appeared** on the Grafana time series graphs. **These spikes were short-lived and quickly corrected as the load balancer re-evaluated the metrics and redirected traffic to less-loaded substations**. The overall trend showed a self-correcting behaviour where each substation alternated between increasing and decreasing loads, maintaining a roughly balanced distribution over time.
- The system demonstrated the ability to dynamically adjust routing based on current conditions, even under continuous and rapid load. **The fluctuations observed were minor and expected in real-time systems** where decisions rely on periodically updated metrics.

3.4. Visual Proof

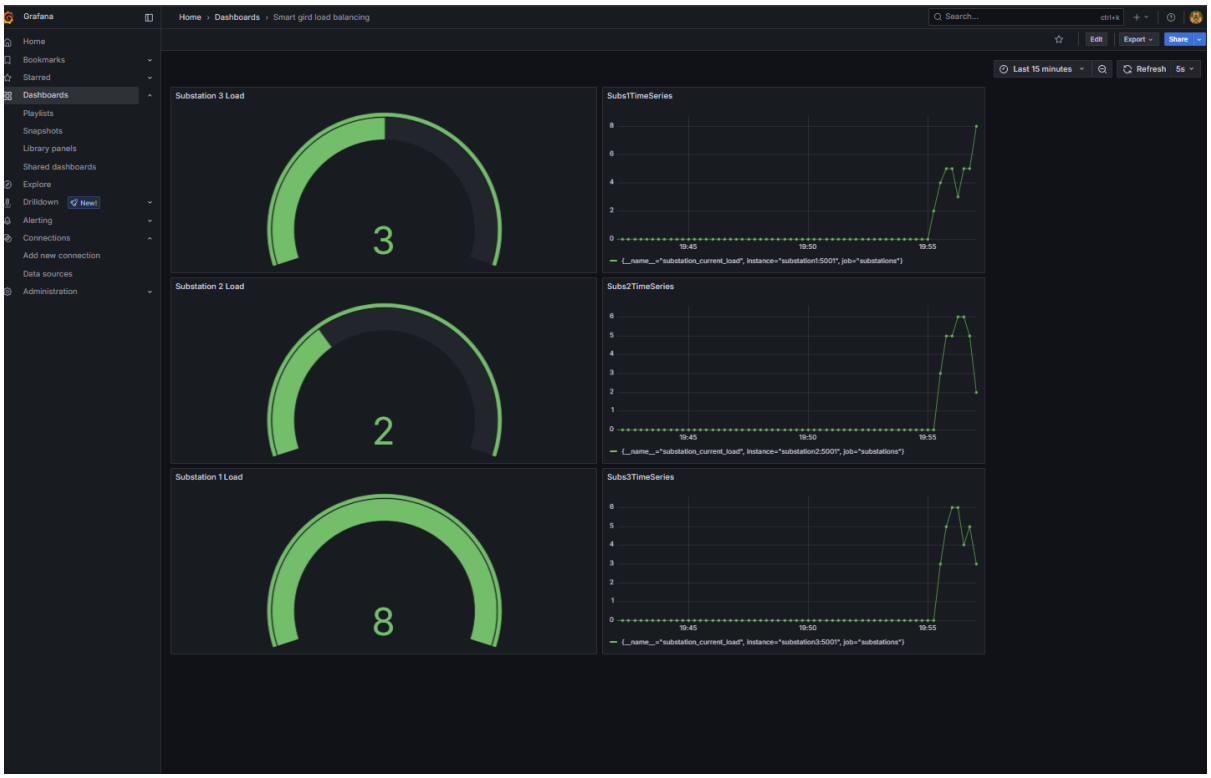
Side-by-side console log of docker instances and test.py running:



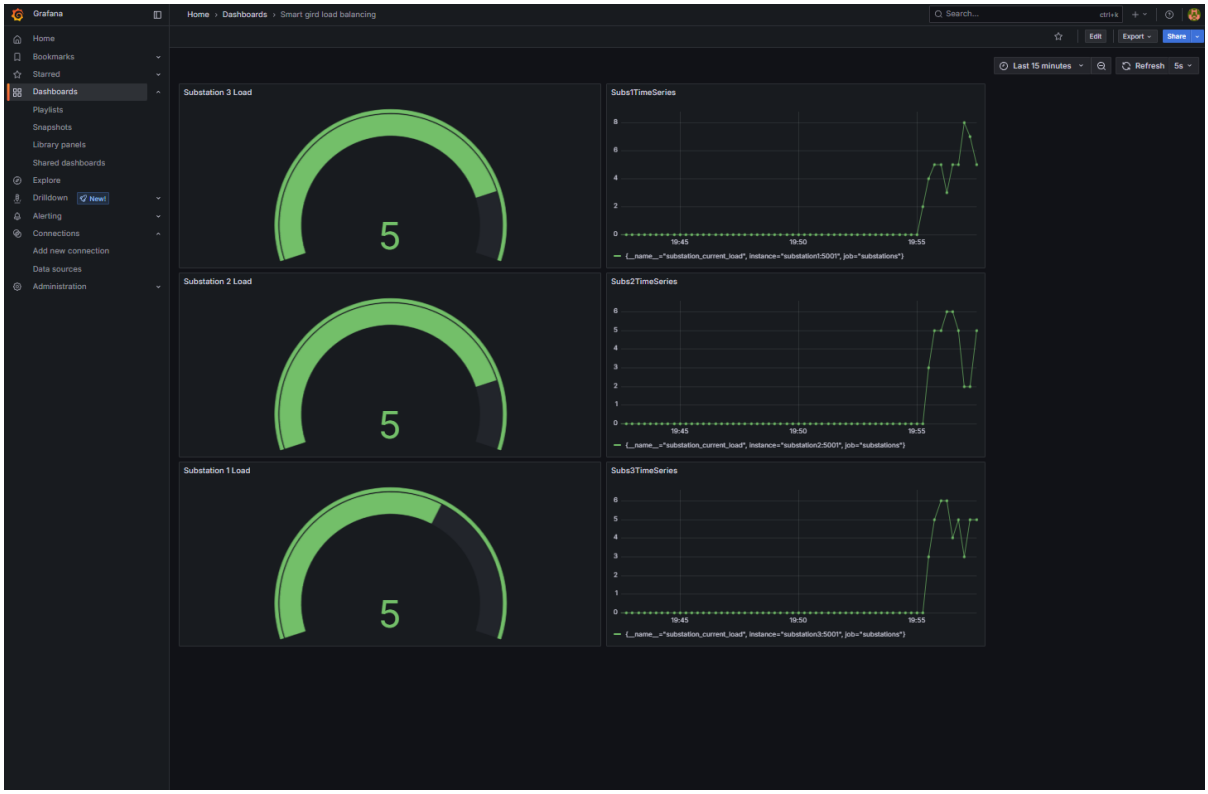
Initial Visualization of Loads on the substations upon testing:



Fluctuation/spikes as load_balancer polls every 5 seconds, a few requests might get routed to the same substation before the load balancer sees an updated metric but soon corrects itself soon after. As this is expected for real-world load balancers.



Few seconds after:



4. Video Demonstration Link

https://youtu.be/Mn2GoVPliDQ