FUNDAMENTALS OF DISTRIBUTED SYSTEMS

Assignment – 1

SUBMITTED BY :
ABHISHEK AGASTI
Roll no. : G24ai2010

Project Report: Dynamic Load Balancing for Smart Grid using Microservices and Observability Stack

Student Details:

- Name: ABHISHEK AGASTI

- Roll Number: g24ai2010

- Course: Fundamentals of Distributed Systems

- Assignment: Assignment 1 – Question 2

Introduction

This project focuses on building a scalable, containerized distributed system for managing electric vehicle (EV) charging using a smart grid approach. With the growing adoption of EVs, efficient and real-time load distribution across multiple substations becomes a critical challenge. This system leverages microservices and observability tools to dynamically assign charging requests to the least loaded substations, ensuring optimal grid utilization and performance.

Objective

To design and implement a microservice-based Smart Grid system that dynamically balances Electric Vehicle (EV) charging requests across multiple substations based on real-time load. The system integrates observability tools (Prometheus and Grafana) to monitor performance.

Technologies Used

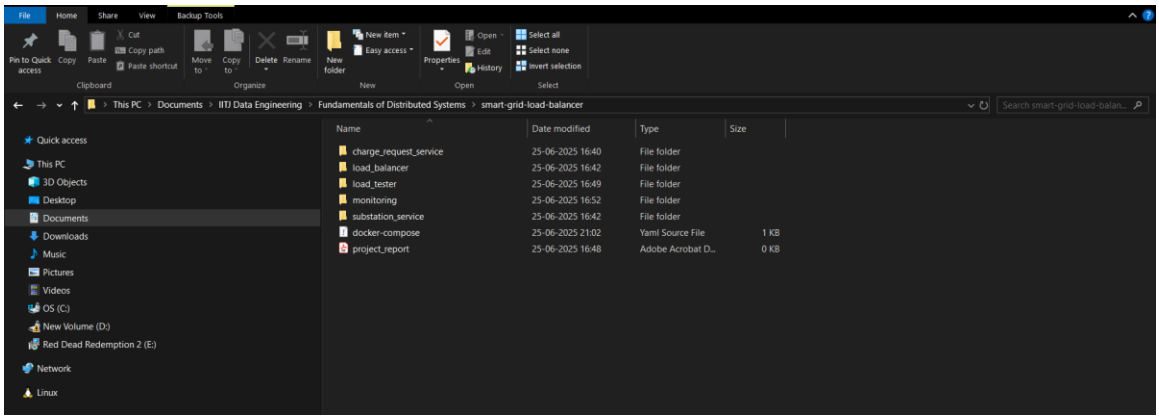
- Programming Language: Python 3.10
- Microservices Framework: Flask
- Metrics Monitoring: Prometheus
- Dashboard Visualization: Grafana
- Containerization: Docker
- Orchestration: Docker Compose

Folder Structure

```
smart-grid-load-balancer/
├── charge_request_service/
│   ├── main.py
```

```
│   └── Dockerfile
├── load_balancer/
│   ├── main.py
│   └── Dockerfile
├── substation_service/
│   ├── main.py
│   └── Dockerfile
├── load_tester/
│   └── test.py
├── monitoring/
│   ├── prometheus/
│   │   └── prometheus.yml
│   └── grafana/
│       └── dashboard.json
├── docker-compose.yml
└── project_report.pdf
```

Screenshot 0: Screenshot showing folder structure of the smart-grid-load-balancer project in the file explorer



Step-by-Step Process

Step 1: Setting up Project Folder

Created all required subdirectories and files using VS Code and command-line utilities.

Step 2: Implementing Substation Service (`substation_service/main.py`)

```
from flask import Flask
from prometheus_client import Gauge, generate_latest, CONTENT_TYPE_LATEST
import threading

app = Flask(__name__)
```

```
current_load = Gauge('current_load', 'Current load on the substation')

@app.route('/charge', methods=['POST'])
def charge():
    current_load.inc()
    threading.Timer(5.0, current_load.dec).start()
    return 'Charging started\n'

@app.route('/metrics')
def metrics():
    return generate_latest(), 200, {'Content-Type': CONTENT_TYPE_LATEST}

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5001)
```

Step 3: Load Balancer Logic (`load_balancer/main.py`)

```
from flask import Flask
import requests
import re

app = Flask(__name__)
SUBSTATIONS = ['substation1:5001', 'substation2:5001']

def get_load(substation):
    try:
        response = requests.get(f'http://{substation}/metrics')
        match = re.search(r'current_load\s+(\d+)', response.text)
        return int(match.group(1)) if match else float('inf')
    except:
        return float('inf')

@app.route('/dispatch', methods=['POST'])
def dispatch():
    loads = {sub: get_load(sub) for sub in SUBSTATIONS}
    best = min(loads, key=loads.get)
    res = requests.post(f'http://{best}/charge')
    return f'Routed to {best}\n'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5002)
```

Step 4: Charge Request Service (`charge_request_service/main.py`)

```
from flask import Flask, request
import requests

app = Flask(__name__)
LOAD_BALANCER_URL = 'http://load_balancer:5002'

@app.route('/request_charge', methods=['POST'])
def request_charge():
    res = requests.post(f'{LOAD_BALANCER_URL}/dispatch')
    return res.text, res.status_code

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Step 5: Load Testing Script (`load_tester/test.py`)

```
import requests
import time

for i in range(20):
    try:
        res = requests.post("http://localhost:5000/request_charge")
        print(f"{i+1}: {res.text.strip()}")
    except Exception as e:
        print(f"Request {i+1} failed: {e}")
    time.sleep(0.5)
```

Step 6: Dockerfiles for Each Service

```
FROM python:3.10
WORKDIR /app
COPY . .
RUN pip install flask requests prometheus_client
CMD ["python", "main.py"]
```

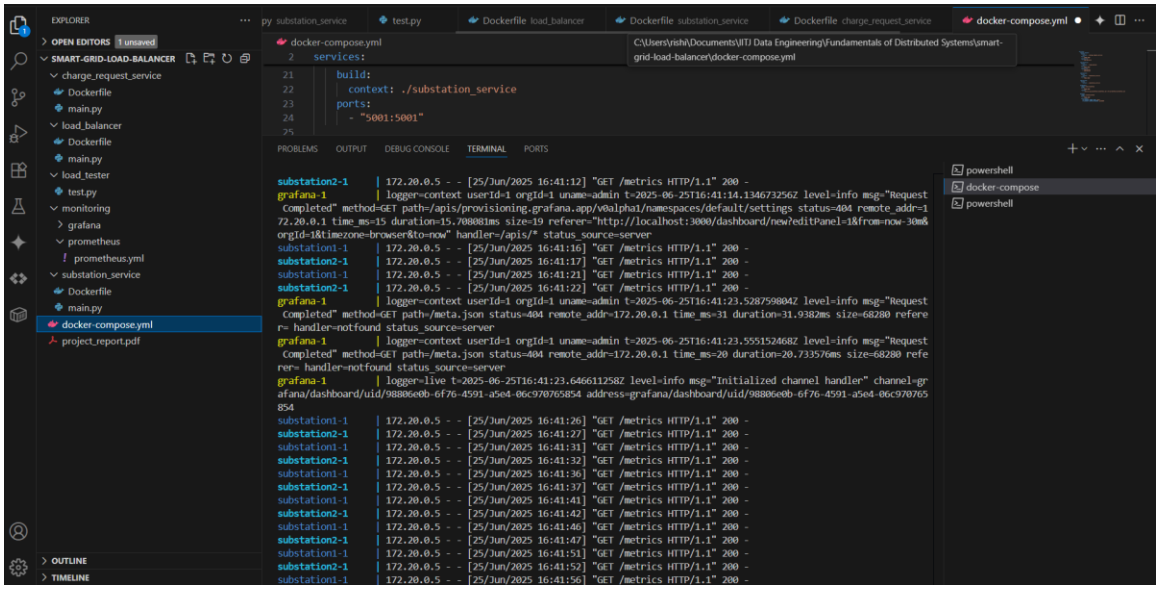
Step 7: Prometheus Configuration (`monitoring/prometheus/prometheus.yml`)

```
global:
  scrape_interval: 5s

scrape_configs:
  - job_name: 'substations'
    static_configs:
      - targets: ['substation1:5001', 'substation2:5001']
```

Step 8: Docker Compose Setup (`docker-compose.yml`)

```
version: '3'

services:
  charge_request:
    build: ./charge_request_service
    ports:
      - "5000:5000"
    depends_on:
      - load_balancer

  load_balancer:
    build: ./load_balancer
    ports:
      - "5002:5002"
    depends_on:
      - substation1
      - substation2

  substation1:
    build: ./substation_service
    ports:
      - "5001:5001"

  substation2:
    build: ./substation_service

  prometheus:
    image: prom/prometheus
    ports:
      - "9090:9090"
    volumes:
      - ./monitoring/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml

  grafana:
    image: grafana/grafana
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=up702094
```

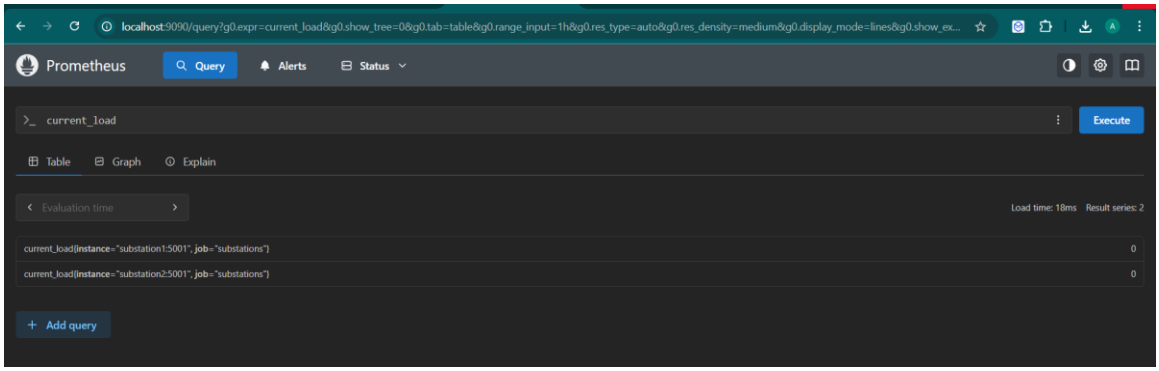
Screenshot 4: Terminal output showing successful build and service startup from docker-compose up --build



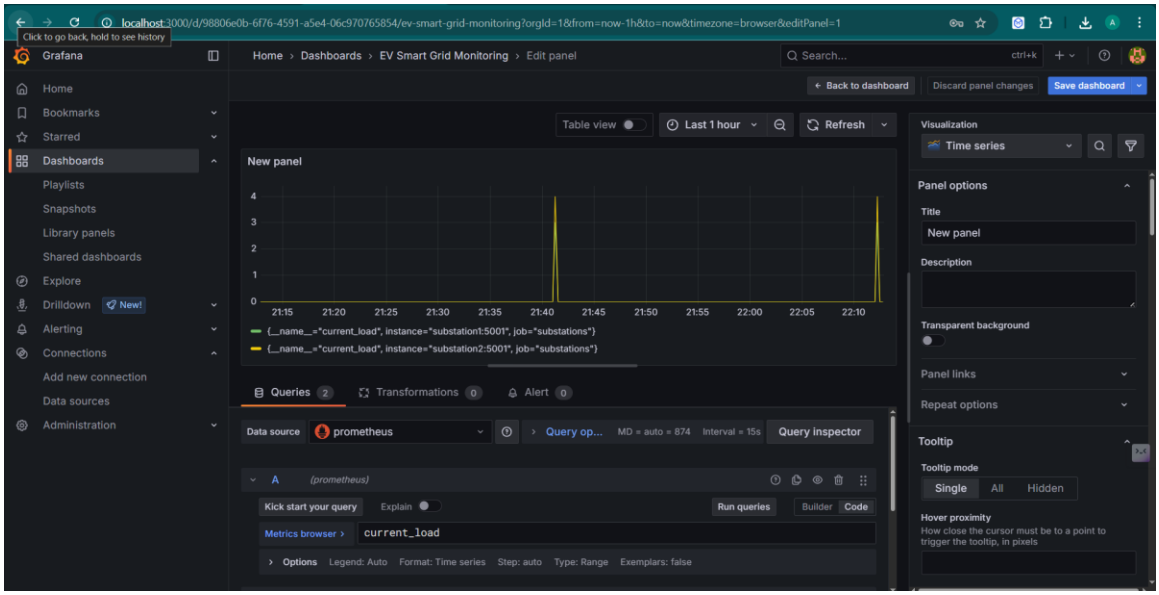
Observability and Monitoring

- Prometheus is used to scrape metrics from each substation service.
- Grafana is configured to show a dashboard with the `current_load` metric.

Screenshot 1: Prometheus http://localhost:9090 showing current_load



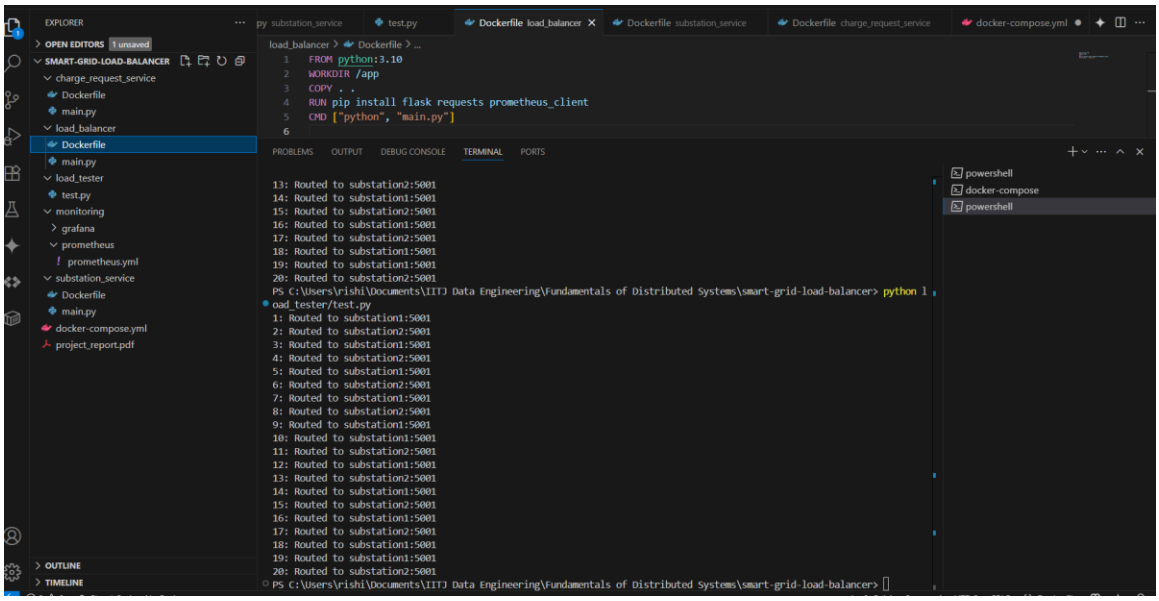
Screenshot 2: Grafana http://localhost:3000 panel displaying substation loads



Load Testing

Load testing simulated via `test.py`
Validates that load is distributed between substations.

Screenshot 3: Terminal output from test.py showing alternating substation routing



Conclusion

This project demonstrates a real-world application of distributed systems principles using microservices and dynamic load balancing. The observability stack allows for effective real-time monitoring.

Video Demonstration

Video Link:

https://youtu.be/AzIucP5t7_Y

https://drive.google.com/file/d/1m1R_HuGmkJUtZFEd1CPTMkueR4Y95dZV/view?usp=sharing

Appendix: How to Run the Project

docker-compose down
docker-compose up --build
python load_tester/test.py