

Objective

The aim of this project is to simulate a Smart Grid system that dynamically balances Electric Vehicle (EV) charging requests across multiple substation services using a custom load balancer, ensuring no substation is overloaded and overall grid health is maintained.

System Architecture

The architecture includes the following services:

- `charge_request_service`: Public API for clients to submit EV charging requests
- `load_balancer`: Routes requests to substations based on real-time load metrics
- `substation_service`: Simulates a substation and exposes its load as Prometheus metrics
- `monitoring`: Prometheus + Grafana to visualize system performance
- `load_tester`: Sends simulated high-volume load

Technology Stack:

- **Language**: Python (Flask, Requests)
- **Containerization**: Docker, Docker Compose
- **Monitoring**: Prometheus (metrics), Grafana (dashboard)

Component Descriptions

Charge Request Service

Receives charging requests and forwards them to the load balancer.

- Validates input
- Calls POST /route on the load balancer

The image shows a VS Code editor with a Dockerfile for the `charge_request_service`. The Dockerfile is located at `smart-grid-load-balancer > charge_request_service > Dockerfile`. The code in the Dockerfile is as follows:

```
1 FROM python:3.9-slim
2 WORKDIR /app
3 COPY requirements.txt .
4 COPY main.py .
5 RUN pip install --no-cache-dir -r requirements.txt
6 RUN pip install flask requests
7 #CMD ["python", "main.py"]
8 ENV FLASK_APP=main.py
9 EXPOSE 5000
10
11 CMD ["flask", "run", "--host=0.0.0.0"]
12
```

The terminal output shows the following logs:

```
charge_request_service-1 | 172.20.0.9 - - [29/Jun/2025 13:24:05] "POST /request-charge HTTP/1.1" 200 -
substation3-1 | 172.20.0.7 - - [29/Jun/2025 13:24:06] "POST /charge HTTP/1.1" 200 -
load_balancer-1 | 172.20.0.8 - - [29/Jun/2025 13:24:06] "POST /route HTTP/1.1" 200 -
charge_request_service-1 | 172.20.0.9 - - [29/Jun/2025 13:24:06] "POST /request-charge HTTP/1.1" 200 -
substation2-1 | 127.0.0.1 - - [29/Jun/2025 13:24:06] "GET /health HTTP/1.1" 200 -
substation3-1 | 127.0.0.1 - - [29/Jun/2025 13:24:07] "GET /health HTTP/1.1" 200 -
substation1-1 | 127.0.0.1 - - [29/Jun/2025 13:24:07] "GET /health HTTP/1.1" 200 -
substation1-1 | 172.20.0.7 - - [29/Jun/2025 13:24:09] "GET /metrics HTTP/1.1" 200 -
substation2-1 | 172.20.0.7 - - [29/Jun/2025 13:24:09] "GET /metrics HTTP/1.1" 200 -
substation3-1 | 172.20.0.7 - - [29/Jun/2025 13:24:09] "GET /metrics HTTP/1.1" 200 -
```

Load Balancer

- Periodically polls all substation /metrics endpoints
- Chooses the least-loaded substation
- Forwards incoming requests using HTTP POST

The image shows a VS Code editor with a Dockerfile for the `load_balancer`. The Dockerfile is located at `smart-grid-load-balancer > load_balancer > Dockerfile`. The code in the Dockerfile is as follows:

```
1 FROM python:3.9-slim
2 WORKDIR /app
3 COPY requirements.txt .
4 COPY main.py .
5 RUN pip install --no-cache-dir -r requirements.txt
6 RUN pip install flask requests
7 CMD ["python", "main.py"]
8 EXPOSE 5000 8000
9
```

The terminal output shows the following logs:

```
substation1-1 | 127.0.0.1 - - [29/Jun/2025 13:26:09] "GET /health HTTP/1.1" 200 -
substation1-1 | 172.20.0.7 - - [29/Jun/2025 13:26:09] "GET /metrics HTTP/1.1" 200 -
substation2-1 | 172.20.0.7 - - [29/Jun/2025 13:26:09] "GET /metrics HTTP/1.1" 200 -
substation3-1 | 172.20.0.7 - - [29/Jun/2025 13:26:09] "GET /metrics HTTP/1.1" 200 -
```

```
smart-grid-load-balancer > load_balancer > main.py > ...
1 from flask import Flask, request, jsonify
2 import requests
3 import threading
4 import time
5 from prometheus_client import start_http_server, Gauge
6 import os
7
8
9 app = Flask(__name__)
10 # Initialize substations list (add before route_request())
11 substations = os.getenv('SUBSTATIONS', '').split(',')
12 print(f"Loaded substations: {substations}") # Debug log
13
14 #substations = [] # Will be populated from environment variables
15
16 # Metrics
17 substation_loads = Gauge('substation_load_percentage', 'Current load percentage', ['substation_id'])
18
19 def poll_substations():
20     while True:
21         for substation in substations:
```

```
substation3-1 | 172.20.0.7 - - [29/Jun/2025 13:26:55] "GET /metrics HTTP/1.1" 200 -
substation1-1 | 172.20.0.7 - - [29/Jun/2025 13:26:55] "POST /charge HTTP/1.1" 200 -
load_balancer-1 | 172.20.0.8 - - [29/Jun/2025 13:26:55] "POST /route HTTP/1.1" 200 -
charge_request_service-1 | 172.20.0.9 - - [29/Jun/2025 13:26:55] "POST /request-charge HTTP/1.1" 200 -
```

Substation Service

- Simulates charging load
- Increments load on new requests
- Exposes /metrics and /health for monitoring

```
smart-grid-load-balancer > substation > Dockerfile > FROM
1 FROM python:3.9-slim
2 WORKDIR /app
3 COPY requirements.txt .
4 COPY main.py .
5 RUN pip install --no-cache-dir -r requirements.txt
6 RUN pip install flask requests
7 RUN apt-get update && apt-get install -y curl
8 CMD ["python", "main.py"]
9 EXPOSE 5000 8000
10
```

```
substation2-1 | 172.20.0.7 - - [29/Jun/2025 13:29:15] "GET /metrics HTTP/1.1" 200 -
substation3-1 | 172.20.0.7 - - [29/Jun/2025 13:29:15] "GET /metrics HTTP/1.1" 200 -
```

The screenshot displays a VS Code editor interface. The Explorer sidebar on the left shows a project structure with folders like 'smart-grid-load-balancer' and 'substation', and files like 'main.py' and 'requirements.txt'. The main editor window shows the code for 'main.py', which is a Flask application. It imports 'Flask', 'request', 'jsonify' from 'flask', and 'start_http_server', 'Gauge', 'Counter' from 'prometheus_client'. It also imports 'random', 'time', 'threading', and 'os'. The code defines a Flask app, sets up Prometheus metrics (Gauge for 'current_load' and 'charging_time', Counter for 'total_requests'), and a health check endpoint at '/health'. The health check function returns a JSON response with 'status': 'healthy'. The bottom panel shows the 'TERMINAL' tab with two log entries from 'substation3-1' and 'substation1-1' showing successful GET requests to the health endpoint.

```
smart-grid-load-balancer > substation > main.py > ...
1 from flask import Flask, request, jsonify
2 from prometheus_client import start_http_server, Gauge, Counter
3 import random
4 import time
5 import threading
6 import os
7
8
9 app = Flask(__name__)
10
11 # Metrics
12 current_load = Gauge('current_load', 'Current number of active charges')
13 total_requests = Counter('total_requests', 'Total charging requests received')
14 charging_time = Gauge('charging_time_seconds', 'Time taken for charging')
15
16 active_charges = 0
17 MAX_CAPACITY = 10 # Max concurrent charges
18
19 @app.route('/health')
20 def health_check():
21     """Simple health check endpoint"""
22     return jsonify({
23         "status": "healthy",
```

substation3-1 | 127.0.0.1 - - [29/Jun/2025 13:29:47] "GET /health HTTP/1.1" 200 -
substation1-1 | 127.0.0.1 - - [29/Jun/2025 13:29:47] "GET /health HTTP/1.1" 200 -

Monitoring & Observability

- **Prometheus** scrapes substation metrics
- **Grafana** visualizes substation loads over time
- Helps observe system behavior during load testing

Docker & Orchestration

- All services are containerized with individual **Dockerfiles** and orchestrated using a single `docker-compose.yml`.
This allows seamless communication between services and independent scaling of substations.

The screenshot shows the VS Code interface with a Dockerfile open in the editor. The Dockerfile contains the following instructions:

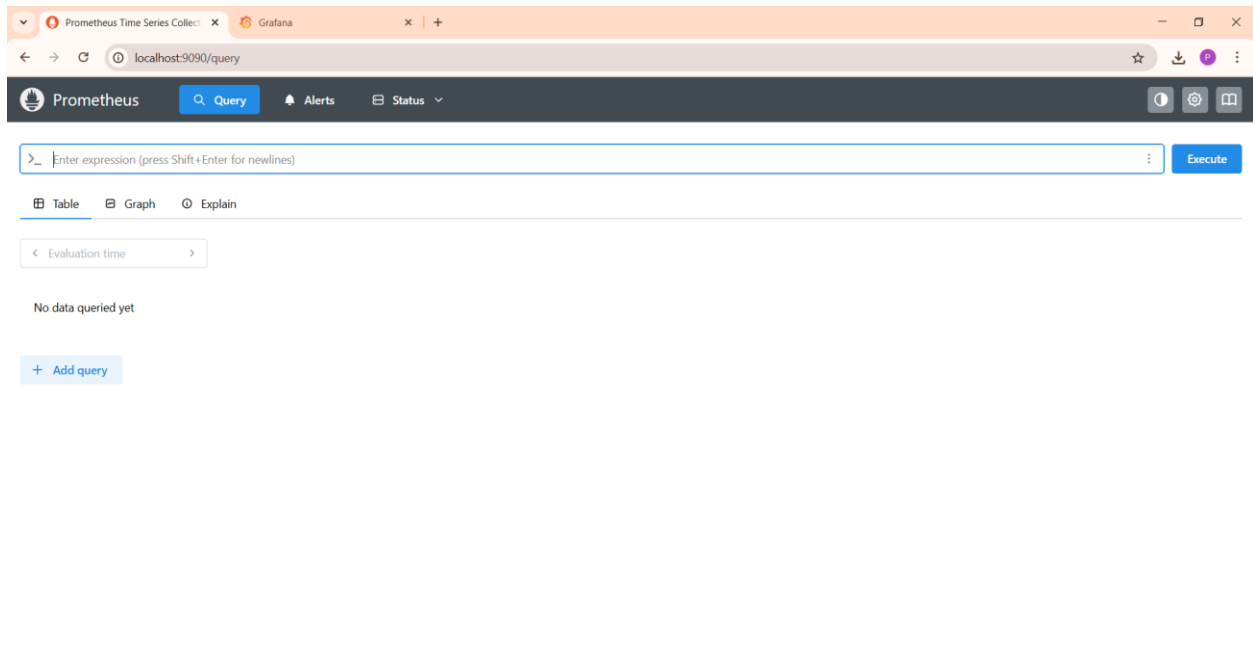
```
1 FROM python:3.9-slim
2 WORKDIR /app
3 COPY requirements.txt .
4 COPY main.py .
5 RUN pip install --no-cache-dir -r requirements.txt
6 RUN pip install flask requests
7 #CMD ["python", "main.py"]
8 ENV FLASK_APP=main.py
9 EXPOSE 5000
10
11 CMD ["flask", "run", "--host=0.0.0.0"]
12
```

The terminal at the bottom shows the output of the docker-compose command, displaying logs for various services including charge_request_service, load_balancer, and substation services. The logs show successful HTTP requests and responses.

Load Testing & Observations

- Load testing simulates multiple vehicle charge requests during a "rush hour" window.
- Prometheus and Grafana show that the load balancer successfully routes requests to the least loaded substations.
- No substation crosses overload threshold.

This screenshot shows a detailed view of the terminal output from the docker-compose command. The logs are color-coded by service and show a sequence of GET and POST requests to various endpoints like /metrics, /health, /route, and /request-charge. The status codes are consistently 200, indicating successful operations.

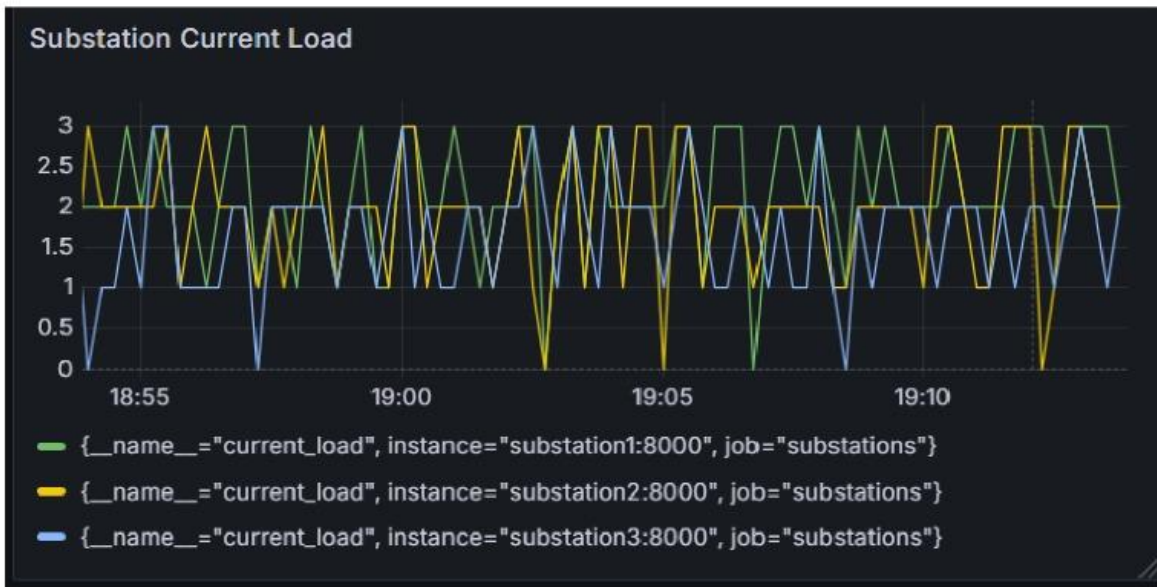


Grafana Dashboard for monitoring load:

Charging Time (in secs)



Substation Load



Conclusion

The Smart Grid load balancing system was successfully implemented using microservices. The system:

- Distributes charging load intelligently

- Exposes real-time metrics
- Demonstrates scalability through service replicas
- Ensures fairness and grid stability under stress