Transport Reliability Analytics

Technical Report

Land Transport Authority (LTA)

Singapore MRT & Bus Network

Stack: Go, gRPC, PostgreSQL, Next.js
Deployment: Docker Compose

October 19, 2025

Contents

1 Executive Summary

This document provides an overview of the Transport Reliability Analytics system - a production-ready full-stack application for tracking and analyzing transport incidents across Singapore's MRT and bus network.

1.1 Quick Start

Start the entire system with one command:

```
docker-compose up
```

Access the dashboard at: http://localhost:3000

2 System Architecture

2.1 Component Diagram

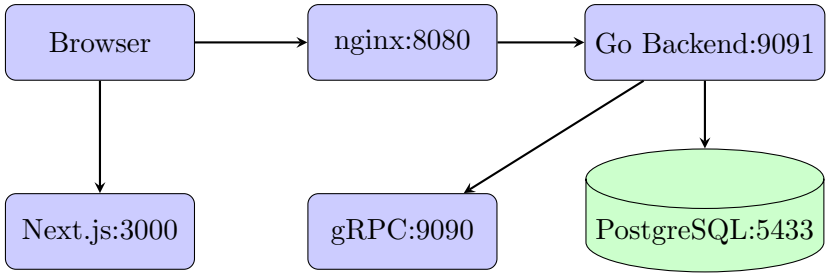

Figure 1: System Architecture

2.2 Technology Stack

Backend: Go 1.24, go-coldbrew, gRPC, PostgreSQL 15
Frontend: Next.js 15, React 18, TypeScript, Tailwind CSS
Infrastructure: Docker, nginx

3 Getting Started

3.1 Prerequisites

- Docker and Docker Compose
- Ports 3000, 8080, 9090, 9091, 5433 available

3.2 Installation

Automated Setup:

```
./setup.sh
```

Manual Setup:

```
docker-compose up
```

3.3 Access Points

Service	URL	Port
Dashboard	http://localhost:3000	3000
API	http://localhost:8080	8080
Backend	http://localhost:9091	9091
Database	localhost:5433	5433

4 API Documentation

4.1 Endpoints

Method	Endpoint	Description
POST	/incidents	Submit new incident
GET	/analytics/top_breakdowns	Top N lines/stations
GET	/analytics/mean_time_between_failures	MTBF calculation
GET	/analytics/recent_disruptions	Recent incidents

4.2 Example: Create Incident

```
curl -X POST http://localhost:8080/incidents \
  -H "Content-Type: application/json" \
  -d '{
    "line": "North South Line",
    "station": "Orchard",
    "timestamp": "2025-10-16T08:32:00Z",
    "duration_minutes": 45,
    "incident_type": "signal"
  }'
```

4.3 Example: Get Top Breakdowns

```
curl "http://localhost:8080/analytics/top_breakdowns?scope=station&limit=5"
```

5 Database Schema

5.1 Tables

lines: id (UUID), name (TEXT), created_at
stations: id (UUID), name (TEXT), line_id (FK), status, created_at
incidents: id (UUID), station_id (FK), line_id (FK), ts, duration_minutes, incident_type, status

5.2 Performance Indexes

- `idx_incidents_ts` - Timestamp descending
- `idx_incidents_line_ts` - Line + timestamp (covering index)
- `idx_incidents_station_ts` - Station + timestamp (covering index)
- `idx_incidents_status` - Status filtering

5.3 Seed Data

- 6 MRT lines (NSL, EWL, CCL, DTL, TEL, NEL)
- 150+ stations
- 400+ incidents (last 90 days)

6 Performance Testing

6.1 Stress Test Results

Two load tests were performed to validate system performance under real conditions:

Light Load Test (10 QPS):

Metric	Result
Total Requests	600
Success Rate	100%
Mean Latency	5.2ms
p50 Latency	5.1ms
p95 Latency	9.4ms
p99 Latency	11.2ms

Heavy Load Test (100 QPS):

Metric	Result
Total Requests	6,000
Success Rate	100%
Mean Latency	3.0ms
p50 Latency	2.6ms
p95 Latency	6.0ms
p99 Latency	8.0ms

6.2 Running Tests

```
# Run all stress tests
make stress-test

# Individual tests
make stress-test-10qps
make stress-test-100qps
```

7 Frontend Dashboard

7.1 Features

- Auto-refresh every 30 seconds
- Interactive bar charts (top breakdowns)
- Color-coded MTBF metrics

- Sortable disruptions table
- Manual refresh button

7.2 Technology

Next.js 15, TypeScript, Tailwind CSS, Recharts

8 Production Considerations

8.1 Scaling

- Horizontal: Multiple app instances behind load balancer
- Database: Read replicas for analytics
- Connection pooling: 25 max, 5 idle

8.2 Monitoring

- Health checks: `/health` and `/ready`
- Structured logging with request IDs
- Prometheus metrics endpoint

8.3 Security

- Change default passwords
- Enable SSL/TLS
- Implement API authentication
- Add rate limiting
- CORS via nginx

9 Troubleshooting

9.1 Common Issues

Port Conflicts: Edit `docker-compose.yml` to change ports

Database Issues:

```
docker-compose logs db
docker-compose exec db psql -U lta_user -d transport_reliability
```

Frontend API Errors:

1. Verify nginx: `docker-compose ps nginx`
2. Test API: `curl http://localhost:8080/health`
3. Hard refresh browser: Cmd+Shift+R

10 Conclusion

The Transport Reliability Analytics system is a production-ready application demonstrating:

- Modern microservices architecture
- High-performance REST and gRPC APIs
- Real-time data visualization
- Professional Docker deployment

10.1 Project Structure

```
.
        backend/          # Go services
        database/         # PostgreSQL setup
        frontend/         # Next.js dashboard
        nginx/            # Reverse proxy
        proto/            # Protocol buffers
        scripts/          # Stress tests
        docker-compose.yml
```

Repository: `stunning-octo-eureka`
Tech Stack: Go · gRPC · PostgreSQL · Next.js · Docker · nginx