# Rail Transportation System - Architectural Design

## Team Information

**Team:** D

**Team Members:**

- Miguel Angel Rubiano
- Martin Erxleben
- Diego Nicolas Ramirez Maldonado
- Jesus Ernesto Quiñones Cely

---

## Software System

**System Name:** Rail Transportation System

**Purpose:**
The Rail Transportation System is responsible for managing and coordinating freight rail operations across Colombia's national railway network. It orchestrates the movement of cargo from ports (Caribbean and Pacific) to inland distribution centers and vice versa. The system integrates real-time fleet tracking, route optimization, cargo inventory management, and IoT-based monitoring of trains, stations, and cargo containers. By providing visibility and control over rail logistics operations, this system ensures efficient, safe, and coordinated cargo transportation as part of the broader national logistics ecosystem.

---

## Model Redesign

### 1. Justification

This process operationalizes the system’s capacity to ensure availability, performance, interoperability, and fault tolerance during peak seasonal demand and adverse operational conditions. The model established a functional baseline capable of coordinating cargo, routes, and fleet operations through distributed services and asynchronous communication. However, under extreme conditions—such as DoS attacks, infrastructure overloads, or multi-system synchronization failures—the SoS exhibited potential degradation points. The redesign explicitly introduces architectural tactics and patterns to guarantee quality attribute goals, ensuring system resilience, adaptability, and reliability.

The redesign justifies the incorporation of new elements like:

- Load balancers, caches, and redundant service instances to improve availability and performance.

- Asynchronous connectors, circuit breakers, and message queues to mitigate cascading failures.

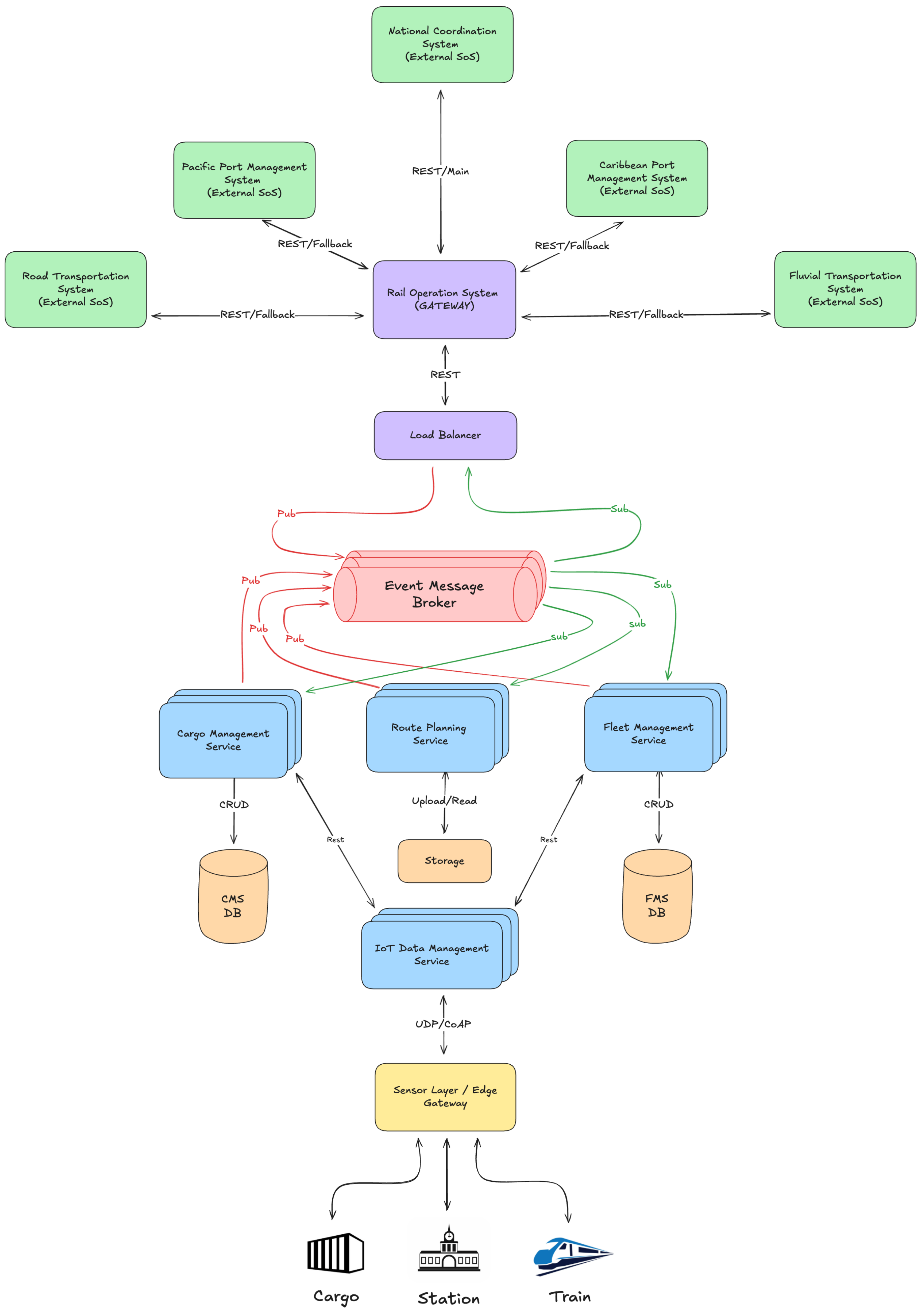
- Security filters and rate limiters to enforce integrity and prevent injection attacks.

- Monitoring and health-check components for observability and self-healing.

---

## 2. Architecture Overview

The following diagram illustrates the complete architectural design of the Rail Transportation System, showing the integration between external SoS systems, the gateway layer, internal services, and physical components:


National Coordination System (External SoS)
REST/Main
Pacific Port Management System (External SoS)
REST/Fallback
Caribbean Port Management System (External SoS)
REST/Fallback
Road Transportation System (External SoS)
REST/Fallback
Rail Operation System (GATEWAY)
REST/Fallback
Fluvial Transportation System (External SoS)
REST
Load Balancer
Pub
Sub
Event Message Broker
Pub
Pub
Pub
Sub
Sub
Sub
Cargo Management Service
Route Planning Service
Fleet Management Service
CRUD
Rest
Upload/Read
Rest
CRUD
CMS DB
Storage
FMS DB
IoT Data Management Service
UDP/CoAP
Sensor Layer / Edge Gateway
Cargo
Station
Train

**Key Components:**

- **External SoS Layer:** National Coordination System, Port Management Systems (Pacific & Caribbean), Road and Fluvial Transportation Systems
- **Gateway Layer:** Rail Operation System (Gateway) and Load Balancer for traffic distribution
- **Event-Driven Core:** Event Message Broker enabling asynchronous communication via Pub/Sub pattern
- **Service Layer:** Cargo Management Service (CMS), Route Planning Service (RPS), Fleet Management Service (FMS), and IoT Data Management Service (IDMS)
- **Data Layer:** Persistent storage (CMS DB, FMS DB) and cache for performance optimization
- **Physical Layer:** Sensor Layer/Edge Gateway connecting trains, stations, and cargo containers

---

## 3. Tactical approach

### 3.1 SoS-level tactics

- **Interoperability and fault tolerance:** To handle overloads and external attacks directed at the National Coordination System (NCS) Gateway, and to manage heterogeneous message formats coming from the ports and subsystems, we incorporated the Gateway, Adapter, and Event-Driven Architecture patterns.

  The gateway is responsible for normalizing input formats (e.g., converting REST messages to AMQP or MQTT), while the event-driven approach allows asynchronous coordination between systems, avoiding bottlenecks.

  Additionally, a Load Balancer distributes the incoming traffic dynamically across instances, maintaining system responsiveness and fault tolerance under high demand or partial failures.

- **Availability:** If a malicious agent tries to use some sort of attack using our API, we implemented filters like input validation, parametrized queries. We also defined some tactics such as rate limiting and network segmentation in order to protect our endpoints.

- **Performance:** During high-demand seasons, the SoS can experience a surge of up to 10,000 messages per second from IoT devices and port systems. To maintain responsiveness, we applied Elastic Scaling (automatic horizontal scaling of broker instances) and Load Shedding (discarding low-priority events).

  Backpressure Control was also implemented so that IoT producers slow down event generation when congestion is detected. Finally, through Observability (Prometheus and Grafana), the system dynamically monitors latency, throughput, and resource usage, ensuring stable performance with latency ≤ 200 ms even under stress.

### 3.2 Subsystem-level tactics

- **Interoperability and fault tolerance:** Our system has to synchronize cargo manifests from pacific and caribbean port while processing multiple NCS real-time requests for route reasignation/update. The gateway helps by transforming the manifest into an unified JSON format, and performs semantic validation before processing. At the same time, the event broker uses priority queues to ensure the consistency across subsystems during overload.

- **Availability:** When one of the Fleet Management Service (FMS) instances fails due to overload, the architecture uses Redundancy (Active-Active), Circuit Breaker, and Health Check patterns.

The Load Balancer detects unresponsive instances through health probes and reroutes traffic, while the orchestrator automatically launches a replacement container in under 60 seconds.

A Message Queue preserves pending events, ensuring that no critical messages are lost during recovery. This guarantees a rapid restoration of service without affecting cargo operations.

- **Performance:** During Christmas, when RTS receives a ton of simultaneous requests and IoT updates, affecting the performance, so, to maintain it, an asynchronous processing is being implemented, at the same time as an auto scaler and a horizontal autoscaler are working.

  Rate limiting protects backend services from overload by throttling excessive client requests.

  The Cache stores frequently accessed routes and train availability data, reducing database queries by over 60%.

  Horizontal scaling automatically deploys new service instances when CPU usage exceeds thresholds, and asynchronous message queues allow non-blocking communication.

---

## 4. Quality traceability

| Quality Attribute | Scenario (from Delivery 3) | Tactic / Pattern Applied | Model Element(s) | Expected Effect / Metric |
|---|---|---|---|---|
| **Interoperability & Fault Tolerance** | Multi-system synchronization failure during transfer — simultaneous manifests from Pacific and Caribbean ports while NCS sends real-time route updates. | Gateway, Adapter, Event-Driven Architecture, Priority Queues, Asynchronous Messaging | `component ros`, `component broker`, `connector amqp`, `connector MQTT`, `component idms` | 100% of manifests normalized to JSON v3.0; synchronization latency < 5s; zero inconsistent assignments between CMS, FMS, RPS. |
| **Availability** | Cascading failure of Fleet Management Service (FMS) during peak operations — one instance crashes under load. | Redundancy (Active-Active), Circuit Breaker, Health Check, Message Queue, Database Replication | `component fms`, `component rts_healthcheck`, `connector health_probe`, `component broker`, `component fms_db` | Recovery < 60s, 0% data loss, system availability ≥ 99.98%, graceful degradation under failure. |

| Quality Attribute | Scenario (from Delivery 3) | Tactic / Pattern Applied | Model Element(s) | Expected Effect / Metric |
|---|---|---|---|---|
| **Performance** | Extreme traffic surge during Christmas peak — 4,300 cargo assignment requests in 10 minutes plus IoT updates. | Load Balancer, Caching, Elastic Scaling, Rate Limiting, Asynchronous Processing, Database Connection Pooling | `component rts_load_balancer`, `component rts_cache`, `component broker`, `component cms`, `component rps`, `component fms` | Latency ≤ 800ms (p95), throughput ≥ 90% of theoretical max, CPU < 80%, no service downtime. |
| **Security / Availability** | Malicious injection attack targeting `/cargo/update` endpoint on Rail Operation Gateway | Input Validation, Parameterized Queries, Network Segmentation, WAF, Rate Limiting | `component ros`, `component cms`, `component cms_db`, `connector RESTful ros->cms` | 100% malicious payloads blocked; no unauthorized DB access; availability ≥ 99.9%. |
| **Observability / Reliability** | Continuous monitoring and recovery of services under load or partial failure | Monitoring Service, Health Checker, Logging Stream, Alerting | `component rts_observability`, `component rts_healthcheck`, `connector logging_stream`, `connector health_probe` | Complete traceability of events and metrics; detection time < 15s; automatic recovery validation. |

## Summary

The Rail Transportation System is architected as a modular, event-driven, and cyber-physical system that balances autonomy with interoperability. Its internal structure integrates software services (for business logic and coordination), hardware gateways (for edge-level data aggregation), and physical components (trains, stations, cargo) into a cohesive operational unit. By leveraging standardized external interfaces and an internal event-driven backbone, the system seamlessly participates in the broader national logistics SoS, ensuring efficient, transparent, and coordinated rail freight operations across Colombia.