DoSA Project - RailPulse 1

Architecting a real-time train location tracking system

Alecu, Victor (s2675889)
Gosa, Vlad (s2755572)
Osinga, Douwe (s2720922)
Vuurens, Bjorn (s2827506)

25th January 2025

Design of Software Architectures (192111332)

CONTENTS

1 Introduction . . . 4

2 Terms and Definitions . . . 4

3 Scope and Environment . . . 5

3.1 Scope . . . 5
3.2 Goals and Functionalities . . . 5
3.3 Related Systems . . . 6
3.4 Trends and Developments . . . 7
3.5 Domain Aspects . . . 9

4 Stakeholders . . . 10

4.1 Key Stakeholders . . . 10
4.2 Other Stakeholders . . . 10
4.3 Onion Classification . . . 10
4.4 Communication Strategies . . . 11

5 Concerns and Drivers . . . 14

5.1 Data Integrity (Q) . . . 14
5.2 Optimising train scheduling (R) . . . 14
5.3 System Reliability (Q) . . . 14
5.4 Fault Tolerance and Error Handling (R) . . . 14
5.5 Minimal System Downtime (Q) . . . 14
5.6 Timely Notification of Failures (Q) . . . 14
5.7 Integration with Existing Systems (R,Q) . . . 15
5.8 Sensitive Data Protection (R,Q) . . . 15
5.9 Reputation and Customer Satisfaction (B) . . . 15
5.10 Product Differentiation (B) . . . 15
5.11 Low-Latency, Real-Time Decision making (Q) . . . 15
5.12 Modularity (Q) . . . 15
5.13 Scalability, Future Growth (B,Q) . . . 15
5.14 Low Resource Utilisation (Cost Savings) (B,D,Q) . . . 16
5.15 Cybersecurity Threat Mitigation (D,R,Q) . . . 16
5.16 Regulatory Compliance (B, R) . . . 16
5.17 Maintaining Market Share (B) . . . 16

6 Quality Attributes and Scenarios . . . 17

6.1 Availability . . . 17
6.2 Performance . . . 18
6.3 Security . . . 20
6.4 Safety . . . 21
6.5 Deployability . . . 23
6.6 Modifiability . . . 24
6.7 Integrability . . . 25
6.8 Usability . . . 27
6.9 Testability . . . 27
6.10 Maintainability . . . 28
6.11 Connecting Quality Attributes and Stakeholders . . . 30
6.12 QA Prioritisation . . . 30
6.13 Motivation for Response measure thresholds . . . 31

7 High-level architecture . . . 32

7.1 Logical View . . . 32
7.2 Architectural Patterns . . . 32
7.3 Data Flow View . . . 35
7.4 Deployment View . . . 36

8 Component Specific Views . . . 38

8.1 Data Ingestion Layer . . . 38

8.2 Data Processing Layer . . . 39
8.3 Data Distribution Layer . . . 40
8.4 Auxiliary Layer . . . 42

9 Related Processes . . . 44
9.1 System Development and Operation - Waterfall . . . 44
9.2 Requirements Elicitation & Design - Agile . . . 45
9.3 System Implementation - Agile . . . 46
9.4 Testing - Agile . . . 46
9.5 Deployment & Maintenance - DevOps . . . 47

10 Risks . . . 48
10.1 Invalid input data . . . 48
10.2 Increased Latency due to Networking . . . 48
10.3 Reliance on third-party systems . . . 48
10.4 Security Concerns . . . 48
10.5 Integration Challenges . . . 48
10.6 Cost Overruns . . . 48

11 Specifications . . . 49

12 Conclusion . . . 50

References . . . 51

A Work distribution . . . 52

B Concerns and Stakeholders tables . . . 53

C Quality Attributes - Utility Tree . . . 55

1 INTRODUCTION

This document serves as a formal description of the software architecture for the RailPulse system, a real-time train monitoring solution designed to modernise European rail networks. This modernisation is of critical importance, driven by the need to support economic growth, enhance travel efficiency, and meet the demands of increasingly interconnected transportation systems. However, achieving these goals requires overcoming significant challenges, such as ensuring the safe and punctual operation of trains, minimising delays, and providing accurate, real-time information to both operational teams and passengers. RailPulse aims to address these challenges by delivering reliable, accurate, and timely train positioning data across national rail networks. The system obtains data from multiple sources (regional dispatch zones, GPS providers, etc.), processes and augments it with additional details such as train composition and speed, and distributes it to key stakeholders, including rail operators, scheduling teams, maintenance teams, and passenger train monitoring applications. This way, Railpulse provides a unified service that supports safe, punctual, and coordinated train operations. Additionally, given the safety-critical nature of the system, it is designed to meet stringent performance, availability, and safety requirements. Delayed or inaccurate data could lead to severe consequences, such as collisions or operational disruptions. In order to mitigate these risks, low-latency data processing, fault tolerance, and robust error-handling mechanisms are of top priority. This way, RailPulse not only supports the safe and efficient operation of rail networks but also strengthens public trust in rail transportation. This architecture document outlines the system's high-level design, key components, and architectural decisions, emphasising principles such as modularity, scalability, and reliability to create a robust foundation for modernising European rail networks and delivering lasting value to all stakeholders.

This document is structured mostly according to ISO 42010:2022 [14], in order to effectively guide the reader through the proposed solution. Section 2 defines key terms and concepts used throughout the document, whereafter Section 3 outlines the scope, along with the relevant technical and business environments that influence the system's architecture. Next, Section 4 describes and prioritises the product's stakeholders, after which their concerns are described and classified in Section 5. Based on these concerns, quality attributes are identified, refined using ASR scenarios, and prioritised in Section 6. The system's high-level architecture, composed of logical, data-flow, and deployment views is described in Section 7. Additionally, this chapter provides an overview of the most important patterns guiding this architecture, along with an analysis of potential alternatives and a motivation for the chosen solution. Following the C4 model, Section 8 further refines the architecture description by outlining the container and component-level views for the subsystems which compose RailPulse. The implementation of this architecture decisions is further contextualised in Section 9 which presents the development processes (e.g., Waterfall, Agile, DevOps) involved in the realisation of the system. Finally, the paper concludes with a discussion of the possible risks influencing RailPulse's development - along with the appropriate mitigation plans - and an overview of the specifications used in the elaboration of this architecture document.

2 TERMS AND DEFINITIONS

- **ASR** - Architecturally Significant Requirement. An ASR represent a requirement which directly impacts design decisions related to the system.
- **RTPS** - RailPulse Train Positioning System.
- **GSM-R** - Global System for Mobile Communications - Railway
- **ERTMS** - European Rail Traffic Management System
- **ETCS** - European Train Control System
- **ETML** - European Train Management Layer
- **GNSS** - Global Navigation Satellite Systems
- **Serverless function** - A programming concept where the programmer only needs to define a function to be ran for a particular API endpoint, where the hosting provider (AWS, Azure, etc.) handles the platform and other infrastructural concerns.
- **Lambda** - An Amazon Web Services-introduced term for a serverless function.
- **RDZ** - Regional Dispatch Zone

3 SCOPE AND ENVIRONMENT

3.1 Scope

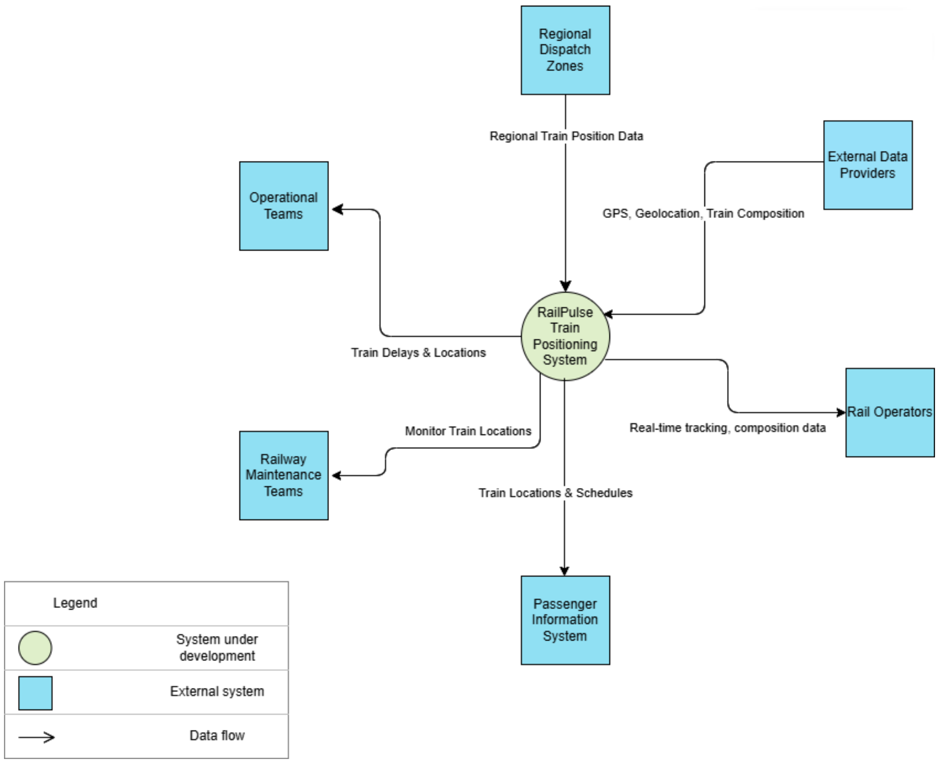

Regional Dispatch Zones
Regional Train Position Data
External Data Providers
GPS, Geolocation, Train Composition
Operational Teams
Train Delays & Locations
RailPulse Train Positioning System
Rail Operators
Real-time tracking, composition data
Monitor Train Locations
Railway Maintenance Teams
Train Locations & Schedules
Passenger Information System
Legend
System under development
External system
Data flow

Figure 1. System context diagram showcasing how external actors interact with the RailPulse system from the context-level viewpoint.

3.1.1 Contextual Overview

The RailPulse system aims to provide a unified national view of train locations, with the goal of enabling safe, punctual and efficient rail services. To achieve this goal, the system relies on external actors such as regional dispatch zones, external data providers and operational teams to consolidate its data. Figure 1 shows a contextual overview of how RailPulse will achieve its proposed goals of offering real-time train tracking services, and other analytics based on the available data. The RailPulse Train Positioning System (RTPS) is presented in the middle of the context diagram. RTPS relies on data flow from three separate external actors. *Regional dispatch zones* offer a local gateway to a nation-wide network by offering train position data in their operational region. *External data providers* provide the system supplemental train position information, such as train composition and geolocation, to ensure precision and redundancy in case that one regional dispatch zone is unavailable. Lastly, *Operational Teams* are both consumers and producers of data from the RTPS. They query train locations, delays and disruptions to provide the system with train schedules based on the conditions in the outside environment. All other external actors are consumers of data from the system. *Rail operators* use RTPS for real-time train tracking, train composition data and scheduling to provide reliable rail services. *Railway Maintenance Teams* query data from RailPulse to ensure safe-working conditions by planning their maintenance during off-peak hours, or when no train will pass the working area for an amount of time. Lastly, *Passenger Information Systems* will use the data provided by RTPS to offer passengers accurate timings, inform them of potential delays and disruptions, and to check train occupancy rates if available.

3.2 Goals and Functionalities

As described by the contextual overview, the system will make heavy use of interfaces to communicate with external actors to gather mission-critical data which is used to provide nationwide visibility which is precise, reliable and fault-tolerant. The main goal of this system is to enhance travel efficiency, both for regular passengers, and for freight operators whose operations rely on timely deliveries of goods. Moreover, it aims to offer a pathway towards the modernisation goal of European rail networks, which is key for economic growth in the area of rail operators and infrastructure. In this section, we offer a brief overview of the functionalities that RailPulse must offer to ensure that the above-mentioned business goals are fulfilled:

1. **Real-time data collection and analysis**. The main functionality of RailPulse is based around train position data, which

is why this functionality stays at the forefront of the system's priority list. The real-time requirement is an essential driver in the architecture of the system.

2. **Data enrichment using supplemental data sources**. By using external data providers, RailPulse can improve the reliability and accuracy of its data.
3. **Nation-wide coverage**. Since the system is meant to offer real-time nation-wide coverage, regional dispatch zones are used as local gateways, which operate across a specific region to ensure low-latency data exchange. These gateways then share regional train positioning data to the central RTPS system, which aggregates it to offer nation-wide coverage.
4. **Fault tolerance**. Since RailPulse is considered a mission-critical system, it is imperative that fault-tolerance is embedded into its design. By making use of multiple data sources, as well as supplemental data, it ensures that at any point in time, if one source of data is unavailable, there must be another one that handles its tasks while maintenance is being done.
5. **Data security and regulation compliance**. The system will comply to regulatory standards such as the European GDPR to ensure all relevant stakeholders of their data integrity, confidentiality and availability.

3.2.1 Constraints and Limitations
To ensure that the system is developed under the proposed budget, it is important to mention the constraints and limitations under which the project is to be developed. In this section, we will present the main constraints and limitations, closely related to a general risk analysis, which we consider that are of importance from the earliest stages of the design process.

1. **Technological Limitations**. Creating new hardware for the purpose of this system is out of scope. The main goal is to use off-the-shelf hardware to enable the functionalities we proposed in the previous section.
2. **Cost and Budget**. To ensure that the system provides the highest possible customer value under specific budget goals, we must prioritise our requirements, and extract all architecturally significant requirements from which functionalities can be prioritised. This architecture will ensure that all significant requirements are documented accordingly, while the other less important ones could be labelled as out of scope.
3. **Regulatory Compliance**. A significant constraint in the architecture of RailPulse is to comply to the rules of EU's GDPR, as well as other security-related requirements that might be imposed by relevant stakeholders. All design decisions must therefore be made through processes such as security-by-design to ensure that the system is compliant to all security regulations.
4. **Fault-tolerance**. Fault tolerance is an important component of the system. This implies that the architectural design of the system must focus on redundancy, an umbrella term which refines into the quality attribute of availability. Since availability is a big-factor of concern for the involved stakeholders, the design decisions related to the functionality of the software must always consider this attribute.

3.3 Related Systems
This section describes related systems, some of which are already present on the market, which have an influence over the final product. Their presence here does not entail that they are going to be integrated with the product, unless it is clearly specified.

3.3.1 European Rail Traffic Management System
The European Rail Traffic Management System [7] is Europe's standardisation effort for the management and interoperation of signalling for railways. It represents an umbrella term for three sub-systems:

- GSM-R - used for communication between trains and railway regulation control centers.
- ETCS - Main system for signalling and control. Sends trackside information to the driver cab using "Eurobalises" (a combination of on-train and trackside equipment).
- ETML - Operation management level used to optimize train schedules.

The main goal of ERTMS is to standardise more than 7 different national systems used by different European rail operators, to ensure EU-wide rail safety, and to take a step towards automatic train operation (ATO). Currently, the system is targeted towards *safety features*, such as railway crossing, train collision and foreign objects on track alerts, *signalling features* such as location estimation through on-train and on-track systems, and *operation management* that optimizes train schedules with the available data. ERTMS represents a good basis for RailPulse's train positioning system since it can use the existing GSM-R and ETCS systems as an external data source.

3.3.2 Robust Train Positioning System
Since Brexit, UK started a collaboration with Thales to modernise their railway network. From that collaboration, the Robust Train Positioning System [31] was created, which aims to provide accurate positioning data and a digital map of UK's railway

network. The system uses an array of on-train sensors that can be retro-fitted to any train that is currently in operation. These sensors collect critical data on the position of the trains using precise accelerometers which keep track of the distance travelled from one point of the track to another. To ensure accurate measurements, this system integrates with the company's existing Train Protection and Warning System (TPWS) which is composed of on-rail sensors, similar to Europe's Eurobalises. At the time of writing this report, this system is still under development.

3.3.3 Train Positioning System using RFID
Intertech Rail, a USA-based company, developed a train positioning system based on RFID technology [10]. The system is based on RFID-tags placed at specific track intervals. Whenever a train passes over one of the tags, the on-train RFID chip passes information to the track sensor, which transmits it to a central control system. The central system then aggregates all the received information to determine each train's location and speed, and to adjust signals and track switches accordingly.

RailPulse integration with ERTMS
As shown in the three related systems above, all major governments are interested in the modernisation of their railway networks through various sensors and systems that increase efficiency and safety. Since RailPulse shares the same goals as Europe's ERTMS, we believe that ETCS and ETML can be used as external data providers, and GSM-R as a reliable communication medium. We will address in subsequent chapters how we aim to use the existing architecture, and improve upon it by integrating newer standards, trends and developments that are described in Section 3.4.

3.4 Trends and Developments
In this chapter, multiple trends and developments in the field of railway location tracking, safety, and software development that enable these services are analysed with the scope of shaping the environment of the RailPulse project. To form a clearer picture of what trends and developments are relevant to our architecture, it has been decided to categorise them in two groups: **follow (F)**, which represent the trends/developments that we are going to implement in this architecture, and **ignore(I)**, which represent the trends/developments that we are not going to use for this architecture.

3.4.1 Hardware-based Trends and Developments
Automatic Train Operation (I) Automatic Train Operation (ATO) is a method of operating trains automatically, without the direct assistance of a human driver. As seen in Figure 2, public interest in the field of autonomous trains started around 2011, most probably due to increased commercial interest in autonomous driving. More and more organisations, including the European Union Agency for Railways, which is the organisation that created ERTMS, are funding research towards improved train tracking and safety systems that could, in the future, enable the possibility for ATOs that fully replace the need of human oversight. The levels of automation are mandated by grades of automation (GoA) in IEC 62290-1. GoA3 represents driverless operation (human assistance might still be needed), and GoA4 which is fully autonomous operation. Our architecture will ignore this trend, since even though the system could provide the necessary data for ATO, this functionality must first be regulated accordingly by governmental regulators.

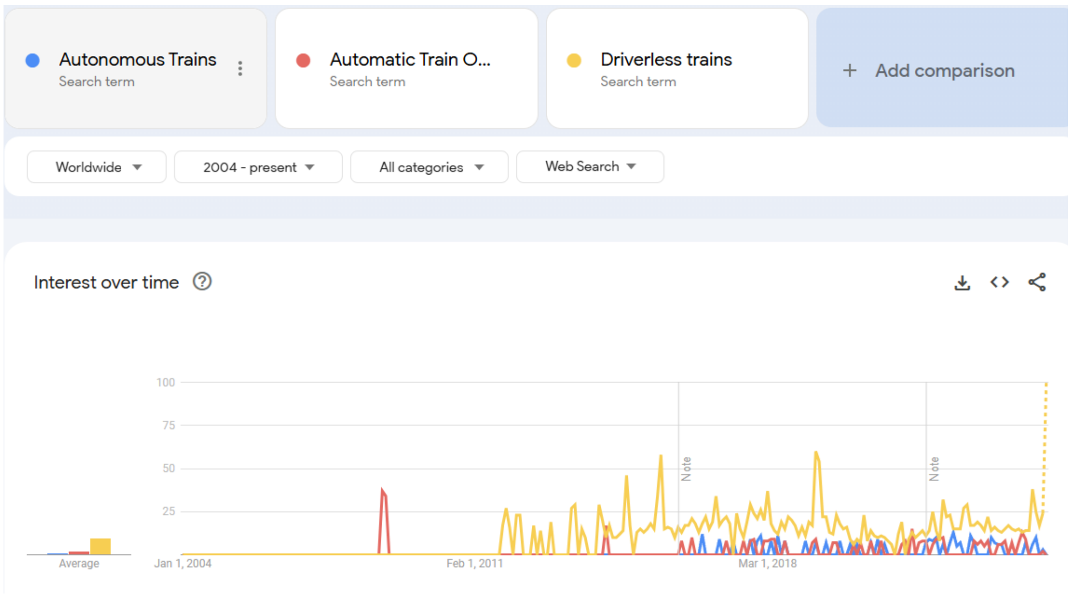


Figure 2. Google Trends snapshot of web search interest in the field of automatic train operation from 2004-01 to 2024-12

ERTMS Adoption by nations outside of Europe (F) According to the ERTMS website [7], more and more countries such as China, India, Taiwan, South Korea and Saudi Arabia are adopting the ERTMS system due to its promising performance, reliability and its advantages regarding maintenance costs.

This is an indication that this system is promising to integrate with, and if the initial EU-focused product is successful, it also potentially opens up the market to countries outside of Europe through the usage of standardised sensors and systems for communication, train positioning and safety. Our architecture will thus follow this trend, since any standardisation effort in terms of the hardware which interacts with the RailPulse system offers improved maintainability, scalability, and potential expandability into other markets.

There is one important limitation to this trend that applies to our system, the security measurements for data processing and storage. RailPulse will integrate with ERTMS, but it will only operate in the EEA region at first, since the only security regulation employed in the system infrastructure is GDPR, which is only guaranteed to cover the rules of the EEA member countries. If the system is to be extended to countries outside of the EEA, a security audit will be performed to verify the system's adherence to the adopter country's data security regulations. If the security audit raises concerns regarding local regulation measurements that are not implemented, RailPulse must integrate them into its architecture before deploying in the adopting country.

Satellite-based train tracking (F) According to Europe's rail research centre, one of the main goals for ERTMS is to reduce or even remove trackside detection elements by replacing them with onboard positioning technologies. This change could positively impact maintenance and deployments costs for positioning systems. One development in the area of onboard positioning is satellite-based positioning systems, based on GNSS. To integrate data from satellite systems into the railway positioning system, the concept of virtual balise-based train positioning is introduced. The virtual balise concept describes the process in which GNSS signals are sent from satellites to trains, which receive the signals through new modules that can be retro-fitted to existing rolling stock. A virtual balise then represents virtual coordinates that the train will reach eventually, and when it does, the train will send its location to a regional dispatch zone. Our architecture will follow this trend, as this technology could provide a trustworthy external data provider source, which could benefit our system's redundancy, reliability and accuracy.

RFID-based tracking (F) In Section 3.3, we discussed an RFID-based positioning system, which uses RFID readers and transmitters to implement physical balises. Whenever a train passes over one balise, its location is sent to a regional dispatch zone which represents a gateway to a national network of train locations. Our system will follow this development since the current ERTMS system already makes uses of Eurobalises, which are radio-based transmitter/receiver systems to determine the locations of trains that pass above them. By incorporating this data source in the RailPulse system, we ensure that the product will be compatible with ERTMS.

3.4.2 Software-based Trends and Developments

Serverless APIs (F) Serverless APIs are a newly-emerging trend of the concept called serverless computing. Instead of deploying your own servers, a third-party cloud provider like AWS or Azure can provide an abstraction layer where the development team only needs to maintain the code, since all the runtime is handled by the provider. Our architecture will use this technology to ensure scalability, maintainability and ease of deployment.

Service-Oriented Architecture (F) Service-oriented architectures represent a method of architecting that uses software components called services to create applications. These services serve a specific business purpose, such as data ingestion, or data processing. The main idea is to loosely couple these services to provide encapsulation, maintainability and scalability. Our architecture will follow this development since it goes hand-in-hand with the serverless API paradigm. Each layer presented in Figure 5 represents one service.

Message Queues (I) Message queues represent a very popular asynchronous communication standard that allows microservices, serverless APIs and even IoT devices to communicate through a producer/consumer pattern. Loosely-coupling modules inside of a system improves efficiency, modifiability and maintainability. However, implementing such an architecture does increase overall code complexity and deployability. Our architecture will not follow this standard since newer, more performant standards exist such as event streaming with Apache Kafka, which promotes on-the-fly data processing.

Event Streaming (F) Event streaming is the practice of capturing real-time data from applications, databases and IoT devices and transporting it to various destinations for immediate processing and storage, or for real-time analysis and analytics reporting. Using event-streaming platforms like Apache Kafka and data-processing libraries such as Apache Spark, our architecture can perform on-the-fly data analysis, processing, validation and serialization for all the necessary consumers down the line.

3.5 Domain Aspects

To help shape the environment within which the software will operate, we define in this section some of the key domain aspects that are relevant for the RailPulse project. The list is not exhaustive, mainly due to the fact that most of the domain aspects are revisited, in more detail, in Section 5 and Section 6. During the initial design stage, most of the domain aspects are extracted based on experience, careful observation of the case study and through the prioritisation of stakeholder concerns. This section only contains two design aspect categories: application and design. The third aspect, realisation, is explained in detail in Section 9.

3.5.1 Application

Safety Safety is an important application aspect of the RailPulse system. Its main purpose is to ensure delay-free, precise data propagation to all the systems that rely on its services. Moreover, since RailPulse handles the data of multiple stakeholders, it must ensure that all sensitive data is protected according to standard data-protection regulations such as GDPR.

Availability and Reliability Availability is a highly-ranked requirement for our stakeholders. First, the system must recover from component failures such as regional dispatch zones going offline. Secondly, the system must operate for extends periods of time without downtime, which can be achieved through redundant data sources and processing nodes.

Cost Focusing on managing and minimising cost ensures that the train tracking system remains efficient, scalable, and profitable, while also aligning with business goals and regulatory requirements. A system which is too expensive to adopt can affect market share, therefore RailPulse must integrate with an existing infrastructure such as ERTMS to manage infrastructure and scalability costs.

Maintainability Maintainability ensures the real-time train tracking system can be upgraded, and adapted to future needs. To achieve this, the system will follow a service-oriented architecture based on microservices. Each microservice can be maintained, upgraded or downright replaced without affecting the overall system since the data flow will be based around a common set of interfaces.

3.5.2 Design

Data integrity Data integrity is relevant for any safety-based system such as RailPulse. To ensure the integrity of data, the system must employ industry standard security practices such as secure channels of communication: HTTPS for REST APIs, hybrid encryption through KEM/DEM for IoT devices that communicate with regional dispatch zones, and salted hashing for user-sensitive data that might be stored in any form of non-volatile storage. Moreover, data integrity can also be impacted through faulty data sources, therefore, yet again, the need for multiple data providers is imperative to verify that the data delivered to all stakeholders is not corrupted.

Recovery Any major faults that could cause system downtime must be covered through fail-safe design. In Section 6.1.2, we expand upon the scenarios and tactics that are employed to ensure that the system has minimal downtime.

Error handling The system must keep track of its state using an auxiliary module at all times. The auxiliary layer is responsible for monitoring the state of the system and for logging all the errors relevant to the involved stakeholders. Fine-grained logging can be used for tracking down problems to individual lines of code causing issues, thus allowing maintenance teams to quickly identify and fix potential errors in the system.

4 STAKEHOLDERS

In this section, we discuss the various stakeholders concerned with the system and motivate the reason for including them. Additionally, we prioritise and classify the stakeholders.

4.1 Key Stakeholders

We have gathered all the key stakeholders in this project and in this section we will discuss who the stakeholders are and why. The key stakeholders can also be found in Table 2.

RailPulse Inc. The company that owns RailPulse is a major stakeholder in this project. They need the product to succeed in order to guarantee financial gains and increased market share.

European Union The European Union (EU) finances the project, due to their interest in enhancing public transportation satisfaction and train freight traffic within Europe, along with boosting the economy.

Rail Operators The rail operators directly manage and operate train services, of which the operational efficiency, scheduling, punctuality, and overall service reliability will be enhanced by the RailPulse system.

Operational Teams (Dispatchers, Controllers) The operational teams are responsible for managing train movements and ensuring smooth operation of the rail network. They are dependent on accurate real-time data to make decisions regarding scheduling, routing and handling unexpected events.

Freight Customers These customers rely on the rail network for the timely and efficient delivery of their goods.

Passengers Passengers will indirectly notice the impact on efficiency, scheduling, punctuality, and overall service reliability that RailPulse offers to its business customers.

Railway Maintenance Team The Railway Maintenance Team is responsible for maintaining the infrastructure and repairing the rolling stock.

Rail Infrastructure Owners This stakeholder manages the infrastructure the rail operators use.

Data Providers (GPS Providers) Data Providers supply the core data that the RailPulse system relies upon, as well as other supplemental data to enrich its services.

Software DevOps Team This team develops and maintain the system, are being paid to ensure the system keeps functioning.

Regulatory Bodies This type of stakeholder has a direct interest in ensuring that the system operates safely, complies with existing legislation, and maintains fair competition, to help the economy and to mitigate security and public safety issues.

Security Team They advise on how to keep the system secure and ensure it adheres to data protection laws such as GDPR.

4.2 Other Stakeholders

The other stakeholders are classified as stakeholders that do not interact with the system directly, but influence its features and behaviour indirectly.

System Architects Design the system, and thus are related to it, making them a stakeholder.

Competitors Competitors will potentially lose income and market share if RailPulse gains popularity, but may also use public information available from RailPulse to better their own services.

4.3 Onion Classification

We have classified all the stakeholders we identified in Section 4.1 into an onion model. This onion model can be seen in Section 4.3 and consists of 3 layers. The inner layer is the affected work area, this layer represents all stakeholders that interact directly with the system. The next layer, the containing organisation consists of all stakeholders within the organisation that control the system but may not interact with it directly. The outer layer, the outside world, contains all stakeholders that are affected by the system but usually do not interact with it. This layer consists of, for example, passengers and the regulatory bodies.

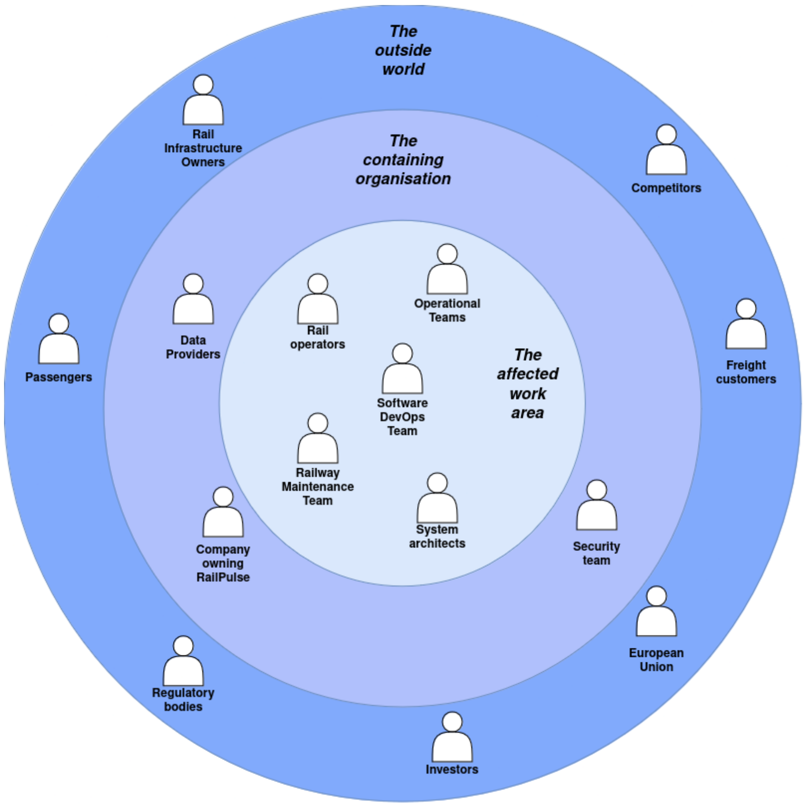


Figure 3. Stakeholder Classification

4.4 Communication Strategies

In this section, we will discuss the strategies we will use to communicate with our stakeholders. To decide which strategies are optimal per stakeholder, we have classified all the stakeholders in a stakeholder-power matrix. This matrix, as seen in Figure 4, helps us visualise which stakeholders we should focus on and which we should keep satisfied.

Moreover, we have created *communication strategy* tables that explain how, about what information, when, and with what method we aim to contact these stakeholders. These tables have been created using the recommended ”minimum required items” from a website called Business Analyst Wiki [23], altered to add one additional row called ”Interaction”. This is the way that we treat the stakeholder: we either keep them informed (”inform”), let them express their opinion about certain system choices but not make design decisions (”think”), or to let them express their opinion *and* make design decisions about parts of the system (”decide”).

Note that competitors are not included in this section due to them being a *negative* stakeholder: whenever this product gains popularity, they lose market share. Thus, we will not inform them of relevant information about the product.

RailPulse inc.

Who	Railpulse Inc.
Interaction	decide
About what	progress on the project, discuss design decisions
When	periodically, every week
With what	meeting

European Union

Who	The European Union
Interaction	think
About what	the progress on the project
When	every month
With what	presentation

Rail Operators

Who	The companies operating on the railway tracks
Interaction	decide
About what	requirements for the system and discuss prototypes
When	every month
With what	meeting or presentation

Rail Infrastructure Owners

Who	The organisation or company responsible for the rail infrastructure.
Interaction	think
About what	safety precautions, relevant QAs of the system (see Table 3)
When	upon reaching milestones that afect relevant QAs
With what	workshop, meetings

Operational Teams (Dispatchers, Controllers)

Who	The teams responsible for managing the train movements and ensuring smooth operation.
Interaction	think
About what	requirements gathering and discussing the system
When	periodically, every month
With what	workshop

Freight Customers

Who	Companies / private entities using freight to move cargo
Interaction	inform
About what	changes to relevant QAs (see Table 3)
When	upon reaching milestones that affect relevant QAs
With what	email

Passengers

Who	People using passenger trains
Interaction	inform
About what	changes to relevant QAs (see Table 3)
When	upon reaching milestones that affect relevant QAs
With what	email

Railway Maintenance Team

Who	the team maintaining the railway infrastructure and rolling stock
Interaction	inform
About what	how to use the system to ensure safety, relevant changes to system (Table 3)
When	upon reaching milestones that affect relevant QAs
With what	workshops, emails

Data Providers (GPS Providers)

Who	the data providers provide all the sensor data for the RailPulse system
Interaction	inform
About what	changes to data collection protocols
When	upon changes to the system which significantly affect data collection.
With what	emails, calls

Software DevOps Team

Who	The team developing and maintaining the system
Interaction	decide
About what	design and implementation decisions
When	every week
With what	meeting

Regulatory Bodies

Who	The government / regulatory bodies
Interaction	inform
About what	changes to privacy/security/other relevant QAs (Table 3)
When	upon major changes to relevant QAs in the system
With what	email

Security Team

Who	The team ensuring the safety and security of the system
Interaction	inform
About what	changes relevant to the security and safety of the system.
When	every 2 weeks
With what	email

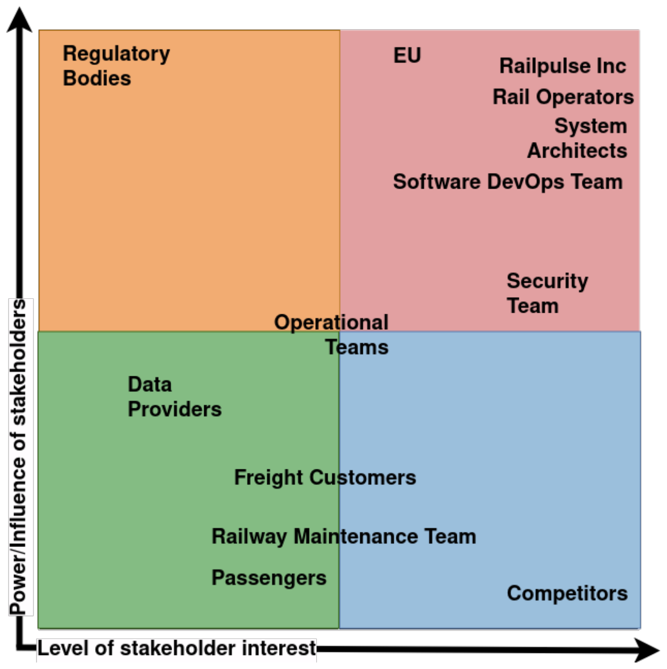


Figure 4. Stakeholder power interest matrix. Interest spans along the x-axis, and power (i.e. influence) scales along the y-axis. Each stakeholder is placed accordingly in the (x,y) plane. The green quadrant represents the *apathetics*, which we must keep an eye on. The yellow quadrant represents the *latents*, which we must keep satisfied. The blue quadrant represents the *defenders*, which we should keep informed. Lastly, the red quadrant represents the *promoters*, which we should keep close at all times.

5 CONCERNS AND DRIVERS

In this section, we list each identified concern, along with a justification and a list of stakeholders who have this concern, along with a justification for their relationship to this concern.

In addition, the concerns of each stakeholder are displayed in Table 3. Each concern is prioritised and classified by a combination of Business Goal (B), Requirement (R), Quality Attribute [4] (Q) (also called ”quality characteristic” [13] or Domain Aspect (D).

5.1 Data Integrity (Q)

We define accurate information as the extent to which the information is true to the original source, i.e., the system should not alter sensor values and should output train locations and such that are sufficiently close to reality for the stakeholders.

This concern is categorised under quality attribute, since it is a characteristic that the system exhibits, and not a particular feature. Moreover, since this is such a general feature of the system, we believe this is a concern for every stakeholder.

5.2 Optimising train scheduling (R)

With optimising train scheduling, we mean that trains should have to make the fewest number of trips between stations, thus travelling the least amount of distance (Accuracy, Efficiency), in order to deliver passengers and freight with acceptable delays (according to the passengers and freight customers). This requirement relates to the quality attributes Accuracy and Efficiency, since train scheduling, if not accurate, can cause higher delays, and the minimisation of the amount of distance travelled per train relates to an efficiency metric.

Since this is one of the main features/requirements of the system, it is categorised under Requirement. The stakeholders are the passengers and freight customers, since they are the ones directly influenced by the train delays resulting from this scheduling, and the rail operators, since customer (dis)satisfaction about scheduling delays will affect their revenue. Moreover, the EU is also interested in this, because this directly affects the effectiveness/efficiency of the new system, which must be great enough to be a successful investment.

5.3 System Reliability (Q)

System reliability is defined in this document as the time that a system has been in use (Maturity), the ratio of time that the system is online (Availability), the extent of the system to deal with unforeseen circumstances / malicious inputs (Fault Tolerance), and the extent to / time in which the system can recover from a fatal crash (Recoverability). These sub-quality attributes are defined in ISO/IEEE 25010-2011 [13].

This is a concern to passengers and freight customers, as the downtime of a system impacts the efficiency of the railway network, and thus the delays of trains with them or their goods on board. This then also impacts operational teams, because they might lose market share over dissatisfied customers. It impacts the company owning Railpulse (Railpulse Inc.) because the operational teams might be (dis)satisfied about the system if it affects their own market share, and it affects the EU because customers will be displeased (at the EU) when the train network is unreliable.

5.4 Fault Tolerance and Error Handling (R)

Fault Tolerance and Error Handling is a specific requirement that arises from the Reliability concern as described in Section 5.3.

While quality system reliability concerns many stakeholders, the specific requirement of fault tolerance and error handling concerns the DevOps team, since they are the ones responsible for making the system fault tolerant.

5.5 Minimal System Downtime (Q)

This is another requirement that is based on the Reliability concern (Section 5.3), which, again, is mainly a quality attribute that concerns the DevOps team as they are responsible for ensuring minimal downtime.

5.6 Timely Notification of Failures (Q)

In order to ensure minimal downtime (Section 5.5), it is of vital importance that when the system experiences a fatal crash, that the DevOps team receives timely notifications. As this talks about a characteristic about certain behaviour of the system, we qualify this under Quality Attribute.

The relevant stakeholder is the DevOps Team, as they are the ones concerned with keeping the system running, and are thus the only direct beneficiaries of this quality attribute.

5.7 Integration with Existing Systems (R,Q)
Integration with other systems, such as the systems of Data Suppliers, is a vital part of Railpulse. Integration can be qualified either as a quality attribute, describing the extent to which the system can work / works with other systems, or as a requirement, mandating a connection to specific data providers.

Since connectivity is vital for the functional part of the application, the quality aspect of it is a concern to all stakeholders of the project, however, the specific connection to data providers can be argued to only be of concern to the DevOps team and the specific Data Provider, since the integration takes data from the specific Data Provider, and needs to be implemented by the DevOps team.

5.8 Sensitive Data Protection (R,Q)
We define sensitive data protection as the extent to which sensitive data is encrypted and/or inaccessible by unauthorised actors (Confidentiality), to which extent the data is not altered and is verified to be of the original source (Integrity, Non-Repudiation, Authenticity), and that information can be traced back to its original creator (Accountability) (terms are from ISO/IEEE 52010 [13]). The act of protecting sensitive data can be seen as a requirement, while the protection/privacy concept is more a behaviour of the system and thus a quality attribute. Sensitive data can be classified as a domain aspect as well, due

Naturally, protection of sensitive data is a concern to the data providers, as some data they share with RailPulse, such as API tokens, should remain confidential. Likewise, Rail operators also share API keys and such and receive operator-specific train information that needs to remain confidential from competitors. If this data leaks out, RailPulse Inc. has a problem, which makes it a concern for them as well. This has indirect effects for all of the other stakeholders, which is why all stakeholders were selected to have this concern.

5.9 Reputation and Customer Satisfaction (B)
Reputation and customer satisfaction aims to capture the feeling of the public and of customers to RailPulse and RailPulse Inc. This is considered as a Business goal, as this does not describe behaviour about or characteristics of the system.

Naturally, RailPulse Inc. is concerned with this, as customer satisfaction directly affects their revenue and reputation. The EU is also concerned with this, as customer satisfaction affects the reputation of the EU as well.

5.10 Product Differentiation (B)
Unlike reputation and customer satisfaction, talked about in Section 5.9, product differentiation does talk about characteristics of the system (namely that it should do something ”different” from other sytsems), but this, again, cannot be translated to system behaviours or characteristics without further inquiry about the state of the market and such, making it best classified as a business goal.

This is of interest to RailPulse Inc., in order to gain market share, and to that extent to competitors, as they get negative value from RailPulse Inc. gaining market share.

5.11 Low-Latency, Real-Time Decision making (Q)
This system concern talks about the latency of different system requirements, from the latency with which train scheduling can be adjusted to the time it takes for RailPulse to generate enhanced location data.

Naturally, since this is talking about the way the system behaves in general, this can be considered a quality attribute. The stakeholders most concerned with this are the ones interacting with the components that require low latency, which are primarily the rail operators, that want to see their scheduling changes appear quickly.

5.12 Modularity (Q)
For the RailPulse software to be more maintainable and more easily extendable, modularity is an important quality of the system. Modularity can be defined as ”degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components” [13, p.14], which is strongly related to modifiability.

The stakeholder most concerned with this is the DevOps team, which will be responsible for extending and maintaining the system, and which are directly affected by the system’s potential (lack of) extendability/maintainability. This modularity is also heavily influenced by the design presented by the architects, which makes them a related stakeholder as well.

5.13 Scalability, Future Growth (B,Q)
We define scalability as the extent to which a system can be expanded, either with new software features, or how effectively it can adapt to processing power per machine (i.e. vertical scaling) or more machines (i.e. horizontal scaling). This is related to

future growth, meaning the extent in which the system will grow in terms of users or features. This is related to modularity as well.

The stakeholder most concerned with future growth is the company itself, since this is a general business goal, as it does not describe behaviour or features of the system, and because future growth is an important part of any company. This then also affects the EU, because a lack of future growth may stunt the potential effectiveness of the EU rail network. However, for future growth, additional features or more computational power may be needed, for which system scalability is a good quality attribute to have (it can be considered a quality attribute as it describes the behaviour of the system).

Since scalability is heavily dependent on architectural design decisions[1], architects are a related stakeholder. Furthermore, the (lack of) scalability of the system directly affects how much effort is needed from the DevOps team to be able to scale horizontally or vertically, or to add features.

5.14 Low Resource Utilisation (Cost Savings) (B,D,Q)

As one can imagine, cost savings does not relate to a behaviour of the system, hence belonging into the category Business goal. System cost savings are usually achieved by using less resources (computational power, memory, bandwidth), which *is* a behaviour of the system, and is thus a quality attribute. However, it is also a system-wide domain concern, as low resource utilisation is a constraint that affects the entire system.

RailPulse Inc., being a for-profit organisation, is naturally interested in saving money, and in order to achieve low resource utilisation, the DevOps team has to do their best to optimise their algorithms and implementations. Furthermore, low resource utilisation is also related to architectural decisions, since an architecture when you would pull for new information constantly will use more resources than an event-based architecture. Therefore the architects are also a related stakeholder.

5.15 Cybersecurity Threat Mitigation (D,R,Q)

Cybersecurity threat mitigation pertain to the ability of the system to defend itself against attacks and its ability to minimise the damages done by an attack, should the attackers be succesfull in bringing down the system or stealing data. This can be concretised into features such as rate limiting (against DoS attacks), encryption (for data protection), among other features.

Since this is a general characteristic of the system, we can consider it a quality attribute. Specific implementations, as mentioned in the previous paragraph, belong more to the category requirements, as they specify singular features that RailPulse must implement. It is also a domain aspect, since security needs to be considered at every step in the architectural process.

The relevant stakeholders for this are architects, as architectural design decisions can affect the level of threat mitigation that is possible, since in a scenario where an API is publically available a lot more threat mitigation might be necessary compared to when it is protected in some way. In addition the DevOps team is also relevant, as they need to maintain these security features, the Security team, as they advise for or against specific measures, and RailPulse Inc. and the EU, as their reputation and capital is at stake.

5.16 Regulatory Compliance (B, R)

Regulatory compliance can be defined as the extent to which the system adheres to regulations imposed by the local authorities or other regulatory bodies, including rules about privacy and environmental sustainability. This is a business goal, as it neither relates to any specific part of the system nor a particular requirement, but it can be defined as specific requirements to ensure the system implementation adheres to the regulations, making it possibly a set of requirements[2].

Naturally, the regulatory bodies are concerned with this, since it is their job to enforce the regulations. Since the EU is partly a regulatory body as well, it is also their concern, and also because violating the regulations would not allow the EU to utilise its own sponsored system.

5.17 Maintaining Market Share (B)

Maintaining market share relates to keeping the existing RailPulse users on the platform, allowing for a stable source of income to support the business and to allow for the spending of financial resources for future growth. This is a business concern, as it does not relate to specific behaviours or requirements of the system.

RailPulse Inc. is concerned with this, as losing market share would influence them directly, costing them money. Furthermore, RailPulse Inc.'s competitors are negative stakeholders in this concern, as RailPulse Inc. maintaining market share would mean that competitors get less customers.

[1]Extending a system with small, interface-based communicating modules will be easier than trying to extend a large monolith

[2]They are in any case (no longer) quality attributes, as "Compliance with standards or regulations that were subcharacteristics in ISO/IEC 9126-1 are now outside the scope of the quality model" [13, p.29].

6 QUALITY ATTRIBUTES AND SCENARIOS

In this section, we identify the most important quality attributes (QAs) for the RailPulse system, refine them into sub-attributes relevant to the problem at hand, and elaborate on scenarios, architectural patterns, and tactics necessary to achieve them. Subsequently, these refined QAs are prioritised via a utility tree, in accordance with the ATAM methodology [15].

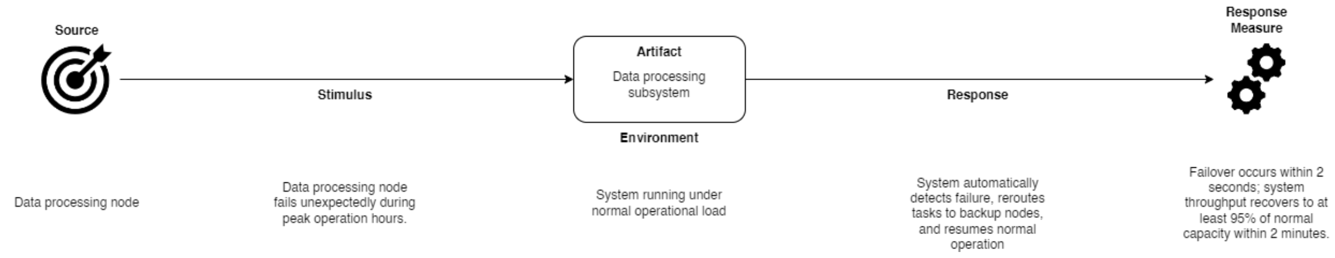
6.1 Availability

Due to the inherent nature and scope of the system, availability is one of the major quality attributes, critical to the success of the project. Downtime can lead to nation-wide delays, enormous amounts of lost capital, scheduling conflicts, and potential safety risks. Ensuring high availability not only has a direct impact on operational efficiency, but can also improve customer satisfaction.

We refine availability into the following sub-attributes:

- **Fault Tolerance**
- **Recoverability**
- **Reliability**

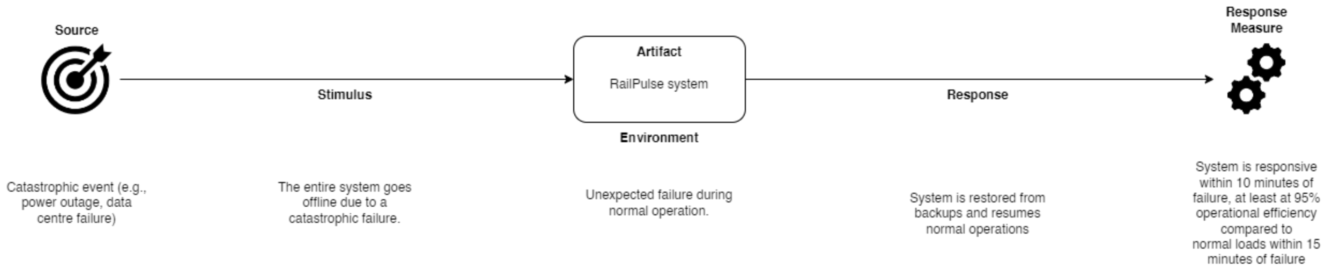
6.1.1 Fault Tolerance - ASR Scenario: System continues operation despite component failures


Source
Stimulus
Artifact
Data processing subsystem
Environment
Response
Response Measure
Data processing node
Data processing node fails unexpectedly during peak operation hours.
System running under normal operational load
System automatically detects failure, reroutes tasks to backup nodes, and resumes normal operation
Failover occurs within 2 seconds; system throughput recovers to at least 95% of normal capacity within 2 minutes.

Business Value: High. This attribute is a must-have for maintaining operational efficiency, which is a mandatory requirement for the overall safety of the system.

Technical Risk: High. Implementing fault tolerance requires careful system design and introduces additional technical overhead. Additionally, extensive testing needs to be conducted to make sure that the fault tolerance functionality works in usual, as well as exceptional, runtime scenarios.

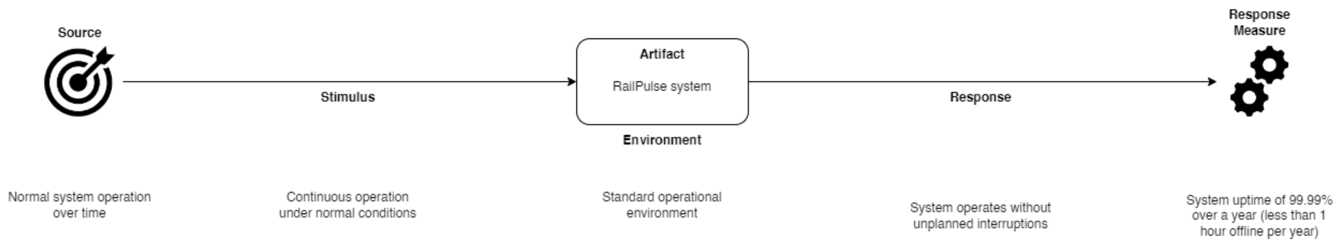
6.1.2 Recoverability - ASR Scenario: System recovers quickly after a complete system outage


Source
Stimulus
Artifact
RailPulse system
Environment
Response
Response Measure
Catastrophic event (e.g., power outage, data centre failure)
The entire system goes offline due to a catastrophic failure.
Unexpected failure during normal operation.
System is restored from backups and resumes normal operations
System is responsive within 10 minutes of failure, at least at 95% operational efficiency compared to normal loads within 15 minutes of failure

Business Value: High. Due to the safety-critical nature of the train location feature, downtime needs to be minimised even for catastrophic scenarios. Stakeholders such as rail operators rely on a constant feed of accurate train locations to prevent risks such as collisions or derailments.

Technical Risk: High. Requires robust backup and recovery mechanisms, which can be complex to implement, test, verify, and maintain.

6.1.3 Reliability - ASR Scenario: System operates without failures over extended periods


Source
Stimulus
Artifact
RailPulse system
Environment
Response
Response Measure
Normal system operation over time
Continuous operation under normal conditions
Standard operational environment
System operates without unplanned interruptions
System uptime of 99.99% over a year (less than 1 hour offline per year)

Business Value: High. Critical for maintaining service levels and contractual obligations.

Technical Risk: Low. Achieving high reliability requires rigorous testing and quality assurance. However, it does not involve complex architectural choices, since it mainly relies on the application of correct coding practices and on hardware quality. Nevertheless, backups for external physical dependencies, such as uninterruptible power sources (UPSs) should be considered in the architecture.

6.1.4 Patterns and Tactics

- **Redundancy**: Redundant components and data sources help eliminate single points of failure.
- **Fault Detection and Recovery**: System health monitoring to detect failures and trigger failover mechanisms automatically (without human intervention) can help response time.
- **Heartbeat Messages / Pinging**: Periodic checking (pinging) to verify the health of system components and data sources allows for automatically detecting failures.

6.1.5 Trade-offs with Other Quality Attributes

- **Performance**: Redundancy and failover mechanisms may introduce additional processing overhead for processes such as the creation and updating of data backups. This could potentially put a high strain on the network, affecting system responsiveness. However, if such operations are planned during periods of low rail traffic[3] the efficiency hit to the system should have minimal consequences.
- **Modifiability and Integrability**: Increased system complexity because of fault-tolerant designs can make future modifications more challenging and time-consuming, as appropriate redundancy/failover mechanisms need to be implemented for every added component, and these redundancy mechanisms need to be integrated with the rest of the system.
- **Energy Efficiency**: Redundant components consume more energy, which may reduce overall energy efficiency and increase operational costs. Additionally, this increased energy use might prove environmentally unsustainable, raising issues in complying with environmental regulations.
- **Cost**: Implementing redundancy and high-availability features requires additional investments in hardware and software, and raises energy costs.
- **Security**: Having backup components, be it hardware or software, requires increased efforts to make sure the system is secure, as each backup service could serve as a gateway for potential breaches, the re-routing to backup components is a potential security issue, and in general, communication is distributed across more endpoints which all need to be monitored and secured.

6.2 Performance

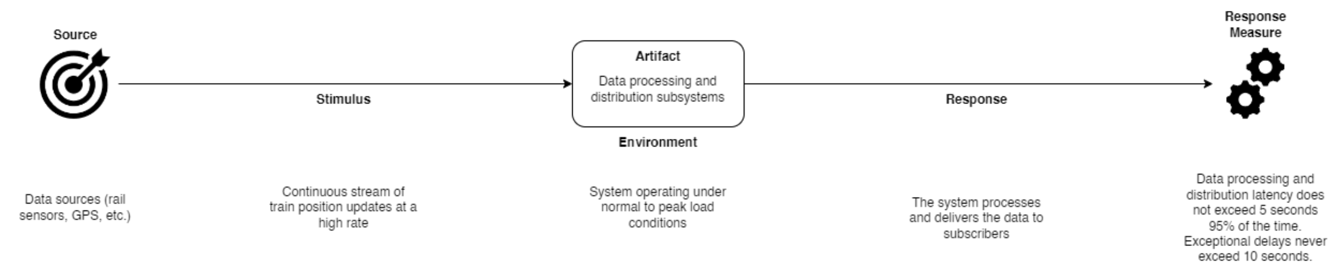
High performance is essential for the system to provide accurate real-time train location data, ensuring the information is processed and delivered swiftly. Without this critical quality attribute, delays in data availability could occur, leading to slower responses to operational events and increasing potential safety risks due to postponed decision-making. Ultimately, performance is essential for adherence with the broader system scope, as well as with important stakeholder concerns.

We refine performance into the following sub-attributes:

- **Latency**
- **Throughput**
- **Scalability**

[3]For example 2-4 AM in the Netherlands for major passenger rail operators [21]

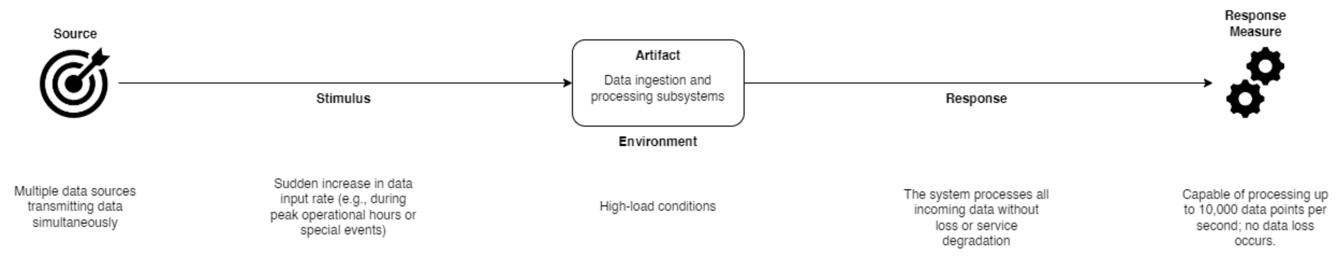
6.2.1 Latency - ASR Scenario: Real-time processing and delivery of train position data


Source
Stimulus
Artifact
Data processing and distribution subsystems
Environment
Response
Response Measure
Data sources (rail sensors, GPS, etc.)
Continuous stream of train position updates at a high rate
System operating under normal to peak load conditions
The system processes and delivers the data to subscribers
Data processing and distribution latency does not exceed 5 seconds 95% of the time. Exceptional delays never exceed 10 seconds.

Business Value: High. Enables timely decision-making by operational teams, maintaining network efficiency and safety. Still, it is not imperative to get extreme levels of low latency, as long as the system can still operate within bounds set by regulatory bodies and stakeholders (an extra second of delay will not hurt the impact of the system as scheduling decisions should not rely on such low margins).

Technical Risk: High. Achieving low latency requires efficient system design and may introduce complexity. Besides, efficient enough hardware needs to be provided for the deployment of the system.

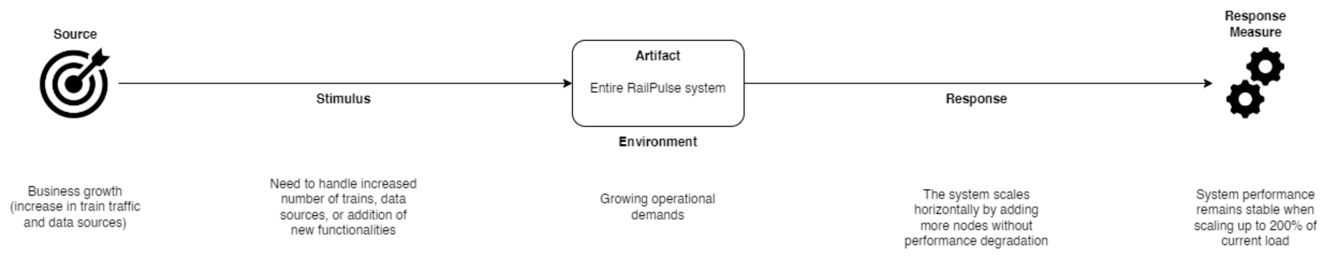
6.2.2 Throughput - ASR Scenario: Handling high volumes of data without degradation


Source
Stimulus
Artifact
Data ingestion and processing subsystems
Environment
Response
Response Measure
Multiple data sources transmitting data simultaneously
Sudden increase in data input rate (e.g., during peak operational hours or special events)
High-load conditions
The system processes all incoming data without loss or service degradation
Capable of processing up to 10,000 data points per second; no data loss occurs.

Business Value: High. Ensures data completeness and timely updates, maintaining service reliability.

Technical Risk: Medium. Requires scalable infrastructure, efficient algorithms, and performant hardware.

6.2.3 Scalability - ASR Scenario: System scales to accommodate increased load and future growth


Source
Stimulus
Artifact
Entire RailPulse system
Environment
Response
Response Measure
Business growth (increase in train traffic and data sources)
Need to handle increased number of trains, data sources, or addition of new functionalities
Growing operational demands
The system scales horizontally by adding more nodes without performance degradation
System performance remains stable when scaling up to 200% of current load

Business Value: Medium. Supports growth and protects investment. However, it is not of critical importance for the immediate future (as the scope highlighted in the case study is still for a national, not continental service).

Technical Risk: Medium. Requires thoughtful architecture decisions in order to future-proof the design and technical patterns involved.

6.2.4 Patterns and Tactics

- **Asynchronous Messaging**: Use message queues to decouple data producers and consumers, reducing latency and increasing system responsiveness.
- **Load Balancing**: Distribute processing tasks across multiple servers to handle high data volumes efficiently.
- **Caching**: Implement caching mechanisms for frequently accessed data to reduce processing time and improve response times.

- **Efficient Resource Management**: Optimise code, utilise efficient algorithms, and employ high-performance computing techniques. These can be validated through enforcement of coding standards and frequent code reviews.
- **Scalable Infrastructure**: Use scalable technologies such as cloud services and container orchestration (e.g., Kubernetes) to handle increased loads seamlessly.
- **Data Partitioning**: Divide data into partitions to allow parallel processing and reduce bottlenecks.

6.2.5 Trade-offs with Other Quality Attributes

- **Modifiability, Integrability, and Testing**: Performance optimisations (load balancing, caching, etc.) may lead to less modular or more complex code, making it harder to modify, extend, and integrate with existing components, possibly making it harder to test as well. Moreover, caching introduces concerns about data validity and cache invalidation.
- **Cost and Energy Efficiency**: High-performance processing may increase energy consumption, affecting operational costs and environmental sustainability.
- **Security**: Performance optimisations might bypass certain security checks in some contexts to reduce latency, potentially opening up the door for vulnerabilities.

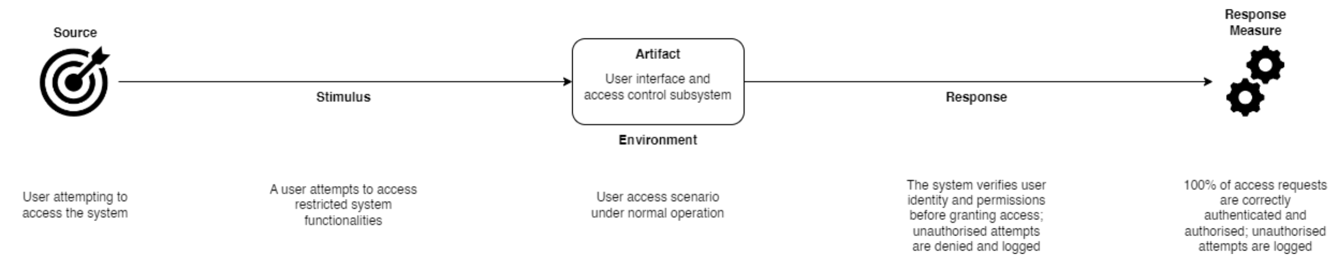
6.3 Security

Security is a critical quality attribute for the RailPulse system due to the sensitive nature of operational data and the potential impact of security breaches. Unauthorised access or manipulation of data can lead to operational disruptions, safety risks, legal liabilities, and loss of public trust. Ensuring robust security measures protects the system from external and internal threats, maintains data confidentiality and integrity, and complies with regulatory requirements.

We refine security into the following sub-attributes:

- **Authentication and Authorisation**
- **Confidentiality**
- **Integrity**
- **Accountability**

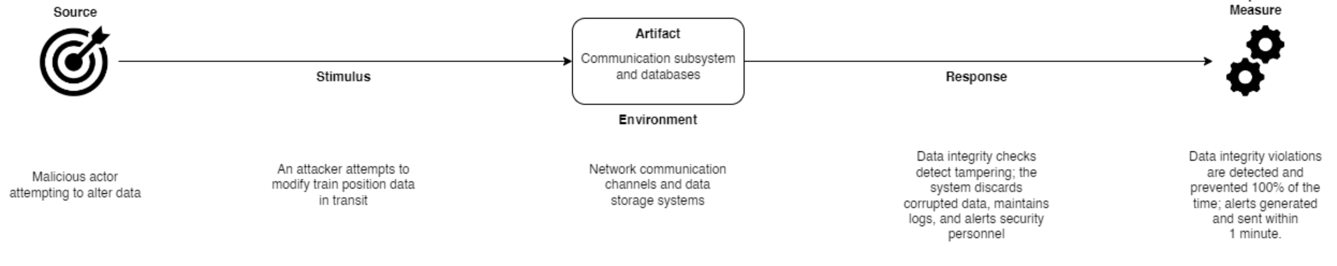
6.3.1 Authentication and Authorisation ($\rightarrow$ Confidentiality) - ASR Scenario : Only authorised users can access system functions and data


Source
Stimulus
Artifact
User interface and access control subsystem
Environment
Response
Response Measure
User attempting to access the system
A user attempts to access restricted system functionalities
User access scenario under normal operation
The system verifies user identity and permissions before granting access; unauthorised attempts are denied and logged
100% of access requests are correctly authenticated and authorised; unauthorised attempts are logged

Business Value: High. Prevents misuse of the system, protects sensitive data, and complies with security policies.

Technical Risk: Low. Requires proper implementation of authentication and authorisation mechanisms, which are not complex operations for an experienced developer. Nevertheless, they should always be thoroughly validated.

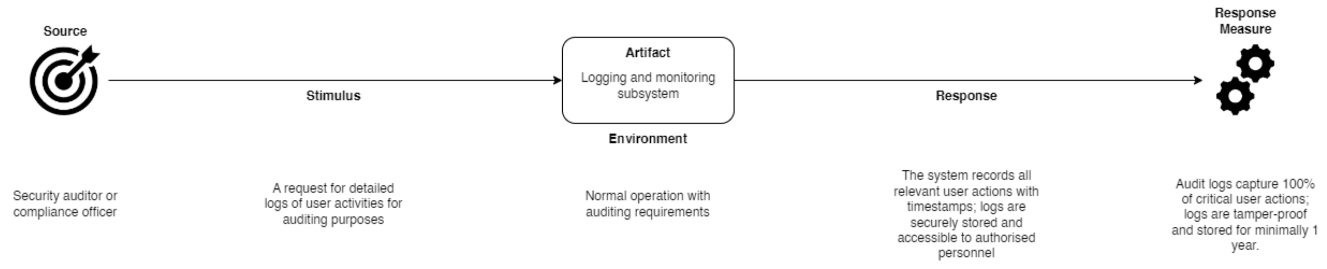
6.3.2 Integrity - ASR Scenario: Ensure data is not tampered with during storage or transmission


Source
Stimulus
Artifact
Communication subsystem and databases
Environment
Response
Response Measure
Malicious actor attempting to alter data
An attacker attempts to modify train position data in transit
Network communication channels and data storage systems
Data integrity checks detect tampering; the system discards corrupted data, maintains logs, and alerts security personnel
Data integrity violations are detected and prevented 100% of the time; alerts generated and sent within 1 minute.

Business Value: High. Ensures reliability of data for operational decisions and prevents potential safety issues.

Technical Risk: Low. Implementing integrity checks relies on established and well-documented industry practices, such as hashing algorithms (e.g., SHA-256) for data integrity verification, digital signatures for ensuring authenticity, and secure transmission protocols like HTTPS. These solutions are mature and widely supported, requiring minimal architectural changes and primarily focusing on correct implementation and configuration.

6.3.3 Accountability - ASR Scenario: All user actions can be traced back to them


Source
Stimulus
Artifact
Logging and monitoring subsystem
Environment
Response
Response Measure
Security auditor or compliance officer
A request for detailed logs of user activities for auditing purposes
Normal operation with auditing requirements
The system records all relevant user actions with timestamps; logs are securely stored and accessible to authorised personnel
Audit logs capture 100% of critical user actions; logs are tamper-proof and stored for minimally 1 year.

Business Value: Low. Supports compliance with possible regulations and internal policies, and aids in incident investigations. Still, inherently the system does not involve much human input and thus possible safety infringements via this avenue are minimal.

Technical Risk: Low. Requires a logging mechanism and proper log management, which are of low technical complexity and don't involve any architectural decision.

6.3.4 Patterns and Tactics

- **Authentication and Authorisation**: Implement strong user authentication mechanisms (e.g., multi-factor authentication) and role-based access control to restrict system access.
- **Encryption**: Use 'stationary' data encryption (e.g., database encryption) and in-transit data encryption (e.g., TLS/SSL) to protect data confidentiality and integrity.
- **Input Validation**: Validate all inputs to prevent SQL injection attacks, cross-site scripting, and other user input-based vulnerabilities.
- **Security Auditing, Logging, and Access Control Lists (ACLs)**: Log security-related events, monitor for suspicious activities, and conduct regular audits. Additionally, define and enforce policies that specify who can access which resources.
- **Firewalls and Intrusion Detection Systems (IDS) / Intrusion Prevention Systems (IPS)**: Protect the system from external threats by monitoring and controlling network traffic.
- **Security Updates and Patch Management**: Keep all system components up to date with the latest security patches and updates, especially if external dependencies are involved.
- **Secure Coding Practices**: Follow best practices to prevent common vulnerabilities (e.g. OWASP Top Ten [24] and NIST Cybersecurity [20]).

6.3.5 Trade-offs with Other Quality Attributes

- **Performance**: Security measures such as encryption and authentication introduce processing overhead, potentially affecting system responsiveness or operational efficiency.
- **Usability**: Strong security mechanisms (e.g., complex passwords, multi-factor authentication) may make the system less user-friendly, potentially impacting usability.
- **Integrability**: Security constraints may complicate integration with external systems, requiring additional development and design effort to ensure compatibility and maintain secure communication.
- **Modifiability**: Implementing security measures can increase system complexity, making future modifications more challenging.
- **Cost**: Enhanced security features require additional resources and possible tool or infrastructure investments.

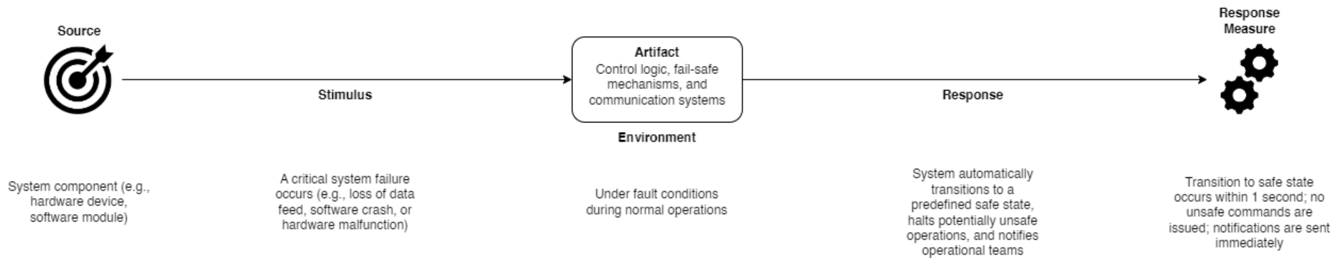
6.4 Safety

Considering the critical domain in which the RailPulse system operates, safety is a critical concern. Issues with the train monitoring module could lead to scheduling mistakes which can have severe consequences, such as accidents with potential loss of life. By ensuring accurate data, implementing safety checks, and providing timely alerts, the system enhances the overall safety of the rail network. Safety considerations may sometimes take precedence over other quality attributes, as the protection of human life and property is paramount.

We refine safety into the following sub-attributes:

- **Fail-safe Operation**
- **Safety Compliance**

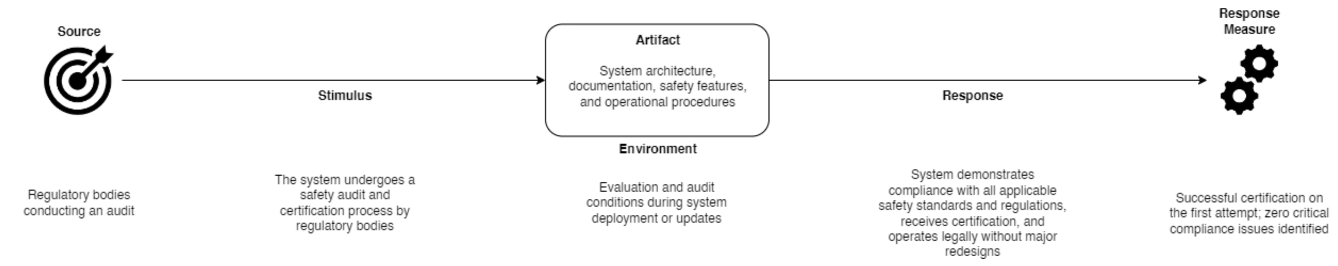
6.4.1 Fail-safe Operation - ASR Scenario: System defaults to a safe state during critical failures


Source
Stimulus
Artifact
Control logic, fail-safe mechanisms, and communication systems
Environment
Response
Response Measure
System component (e.g., hardware device, software module)
A critical system failure occurs (e.g., loss of data feed, software crash, or hardware malfunction)
Under fault conditions during normal operations
System automatically transitions to a predefined safe state, halts potentially unsafe operations, and notifies operational teams
Transition to safe state occurs within 1 second; no unsafe commands are issued; notifications are sent immediately

Business Value: High. Fail-safe operation ensures that the system can handle unexpected failures without causing harm to users, data, or equipment.

Technical Risk: High. Implementing fail-safe mechanisms requires a clear understanding of potential failure modes and robust design strategies to ensure safe transitions. It also involves extensive testing across diverse scenarios.

6.4.2 Safety Compliance - ASR Scenario: System meets all relevant safety standards and regulations


Source
Stimulus
Artifact
System architecture, documentation, safety features, and operational procedures
Environment
Response
Response Measure
Regulatory bodies conducting an audit
The system undergoes a safety audit and certification process by regulatory bodies
Evaluation and audit conditions during system deployment or updates
System demonstrates compliance with all applicable safety standards and regulations, receives certification, and operates legally without major redesigns
Successful certification on the first attempt; zero critical compliance issues identified

Business Value: High. Necessary for legal operation, avoiding fines, and maintaining public confidence.

Technical Risk: Low. Compliance requires comprehensive documentation and adherence to stringent standards, however, these requirements are often well-defined and are mostly already supported by existing frameworks, libraries, and are often integrated into best-practices. Adhering to regulations mainly involves following established processes and ensuring thorough validation.

6.4.3 Patterns and Tactics

- **Design for Safety**: Incorporate safety considerations into the system design from the start, implementing safety checks and validation to prevent hazardous actions.
- **Input Validation and Integrity Checks**: Ensure all input data is accurate, complete, and within expected parameters before processing to prevent undefined behaviour.
- **Robust Error Handling**: Implement error detection and handling mechanisms to prevent the system from entering unsafe states due to exceptions or faults.
- **Redundancy for Safety-Critical Components**: Have backups for critical system components and pathways to reduce the likelihood of safety-affecting failures.
- **Fail-safe Defaults**: Configure the default system parameters to safe values in the event of (human) failures, unexpected inputs, or unexpected conditions.
- **Continuous Monitoring**: Employ real-time system health monitoring, performance metrics, and operational parameters to detect and address anomalies.
- **Safety Audits and Testing**: Conduct regular safety audits, simulations, and rigorous testing (including fault injection and stress testing) to validate safety mechanisms.

6.4.4 *Trade-offs with Other Quality Attributes*

- **Performance**: Implementing safety checks and validations introduces additional processing overhead, which can affect system responsiveness.
- **Complexity**: Incorporating robust safety mechanisms increases system complexity, which may affect maintainability and extend development time.
- **Modifiability**: Changes to safety-critical components necessitate thorough testing and recertification, making modifications more time-consuming and costly.
- **Cost**: Investing in safety features, compliance efforts, and rigorous testing increases development and operational costs.
- **Energy Efficiency**: Redundant systems and continuous monitoring may consume more energy, impacting the system's energy efficiency.

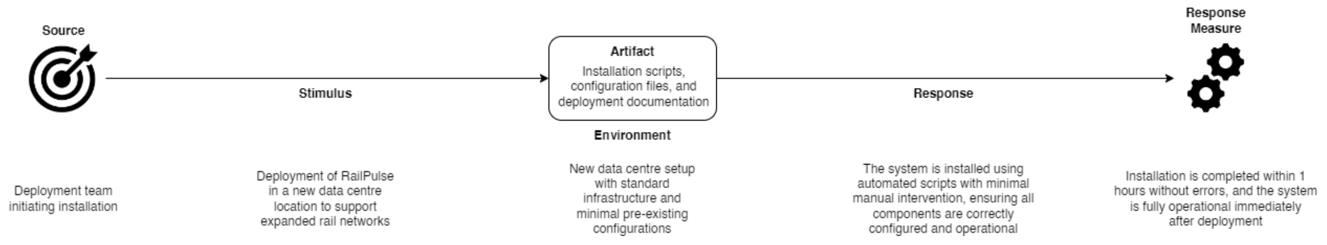
6.5 Deployability

Deployability is also a notable quality attribute of the RailPulse system, ensuring that the system can be efficiently deployed, updated, and maintained in various operational environments. High deployability reduces the time and effort required to install and configure the system, facilitates seamless updates and patches, and minimises downtime during deployments. By focusing on this QA, the system can achieve rapid deployment cycles, adapt to changing requirements, and maintain operational continuity, thereby enhancing overall system reliability and user satisfaction.

We refine deployability into the following sub-attributes:

- **Installability**
- **Upgradeability**

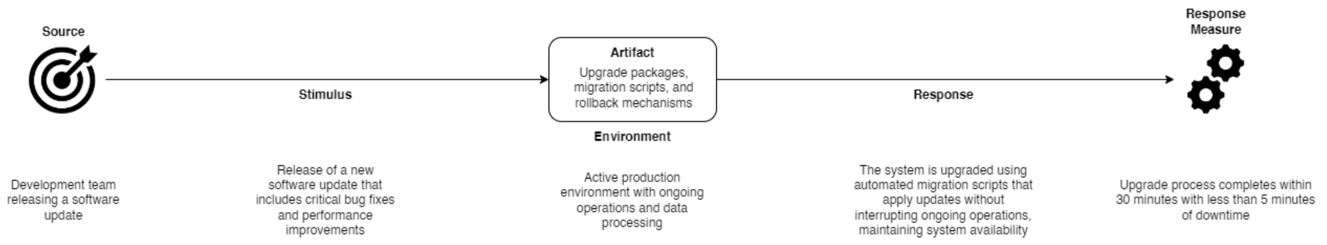
6.5.1 *Installability - ASR Scenario: Efficient installation of the RailPulse system in new data centres*


Source
Stimulus
Artifact
Installation scripts, configuration files, and deployment documentation
Environment
Response
Response Measure
Deployment team initiating installation
Deployment of RailPulse in a new data centre location to support expanded rail networks
New data centre setup with standard infrastructure and minimal pre-existing configurations
The system is installed using automated scripts with minimal manual intervention, ensuring all components are correctly configured and operational
Installation is completed within 1 hours without errors, and the system is fully operational immediately after deployment

Business Value: Medium. Enables rapid expansion and deployment of the system to new locations, supporting business growth and operational scalability. However, a more time-consuming or complex installation process would not critically impact the system's operation, as installations are infrequent and not part of day-to-day activities.

Technical Risk: Low. Ensuring installability mainly requires well-documented and extensively tested installation scripts, which can rely on widely used deployment tools and practices.

6.5.2 *Upgradability - ASR Scenario: Seamless system upgrades with minimal downtime*


Source
Stimulus
Artifact
Upgrade packages, migration scripts, and rollback mechanisms
Environment
Response
Response Measure
Development team releasing a software update
Release of a new software update that includes critical bug fixes and performance improvements
Active production environment with ongoing operations and data processing
The system is upgraded using automated migration scripts that apply updates without interrupting ongoing operations, maintaining system availability
Upgrade process completes within 30 minutes with less than 5 minutes of downtime

Business Value: High. Software updates and upgrades are critical for maintaining the system's reliability, security, and performance. Additionally, they might be a formal requirement from regulatory bodies.

Technical Risk: Low. Implementing upgradability typically involves standard practices, such as version control, modular updates, and rollback mechanisms, all of which are well-supported by existing tools and frameworks. Testing upgrade paths and ensuring backward compatibility further reduces risks during deployment.

6.5.3 Patterns and Tactics

- **Automated Deployment Pipelines**: Utilise CI/CD pipelines to automate the deployment process, ensuring consistency and reducing manual errors.
- **Containerisation**: Deploy the system using containers (e.g., Docker) to ensure consistent environments across different deployment targets.
- **Declarative deployment configurations**: Utilise declarative scripts with tools that support declarative deployment (such as Kubernetes [17]) to make deployment independent of the order of running commands, and let the container orchestrator handle the deployment as much as possible.
- **Blue-Green Deployments**: Implement blue-green deployment strategies to minimise downtime and reduce deployment risks by maintaining two production environments.
- **Rollback Mechanisms**: Design deployment processes with the ability to quickly rollback to previous stable versions in case of deployment failures.
- **Immutable Infrastructure**: Adopt immutable infrastructure principles where servers or services are never modified after deployment, promoting consistency and reliability.

6.5.4 Trade-offs with Other Quality Attributes

- **Security**: Automated and rapid deployment processes may increase exposure to security vulnerabilities if not properly secured. Ensuring secure pipelines and access controls is essential.
- **Performance**: Containerisation and additional deployment layers can introduce performance overheads, though the deployment benefits usually weigh up to this overhead (from experience).
- **Complexity**: Implementing advanced deployment strategies (e.g., blue-green deployments, Infrastructure as Code (IaC)) can increase system complexity and require specialised knowledge.
- **Cost**: Utilising container orchestration platforms and automated deployment tools may incur additional infrastructure-related costs.

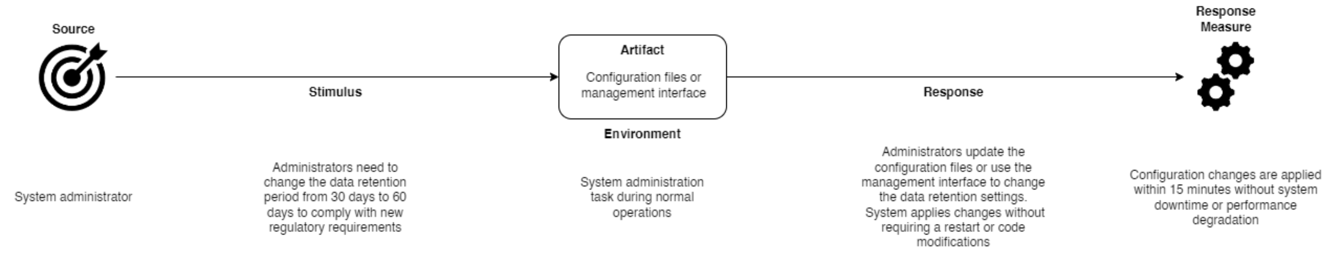
6.6 Modifiability

Assuming that the system promoted by RailPulse is intended as a long-term solution to the railway monitoring problem, a necessary characteristic would be its adaptability to evolving requirements, technologies, and operational needs. The ability to modify the system efficiently reduces long-term operational costs, extends the system's lifespan, and allows for the integration of new functionalities without significant disruptions. By designing the system with modifiability in mind, future enhancements—such as adding new data sources or updating components—can be implemented with minimal impact on existing functionalities.

We refine modifiability into the following sub-attributes:

- **Configurability**
- **Extensibility**

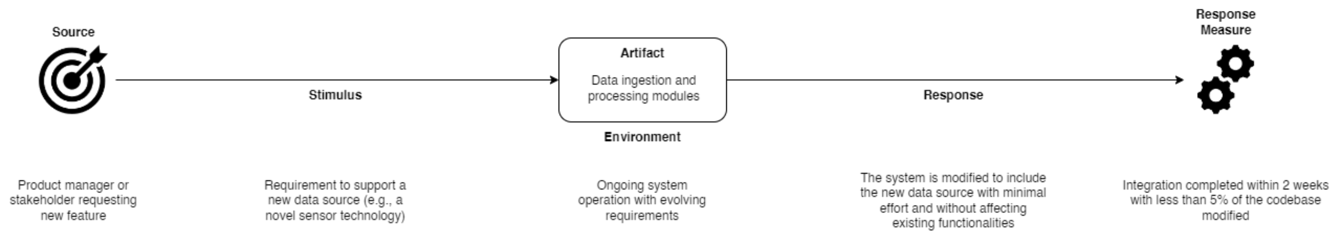
6.6.1 Configurability - ASR Scenario: Adjust system parameters through configuration without code changes


Source
Stimulus
Artifact
Configuration files or management interface
Environment
Response
Response Measure
System administrator
Administrators need to change the data retention period from 30 days to 60 days to comply with new regulatory requirements
System administration task during normal operations
Administrators update the configuration files or use the management interface to change the data retention settings. System applies changes without requiring a restart or code modifications
Configuration changes are applied within 15 minutes without system downtime or performance degradation

Business Value: Medium. Enhances system flexibility and responsiveness to changing operational needs and regulatory requirements.

Technical Risk: Low to Medium. Configuration management is a standard practice and typically involves minimal risk when properly implemented. The main challenge is to identify the most important parameters which (might) need configuration (in the future) and make them adjustable.

6.6.2 Extensibility - ASR Scenario: Adding new data sources to the system


Source
Stimulus
Artifact
Data ingestion and processing modules
Environment
Response
Response Measure
Product manager or stakeholder requesting new feature
Requirement to support a new data source (e.g., a novel sensor technology)
Ongoing system operation with evolving requirements
The system is modified to include the new data source with minimal effort and without affecting existing functionalities
Integration completed within 2 weeks with less than 5% of the codebase modified

Business Value: High. Extensibility is critical for adapting the system to evolving business needs, such as incorporating additional data sources to improve decision-making, analytics, or functionality.

Technical Risk: Medium. Achieving extensibility requires a flexible architecture capable of accommodating new components or data sources without significant rework. This often involves complex architectural decisions, such as designing modular components, implementing well-defined and standardised interfaces, and managing dependencies. While such practices are well-known, ensuring compatibility and scalability across multiple extensions adds additional challenges.

6.6.3 Patterns and Tactics

- **Modular Architecture and Plug-in Components**: Use of modular, service-oriented architecture to encapsulate components, allowing independent development and updates. Also, implement data sources and functionalities as plug-ins or modules that can be added, removed, or replaced without affecting the core system.
- **Abstraction and Interfaces**: Define clear interfaces and abstract data sources to decouple components and facilitate integration.
- **Layered Architecture**: Organise the system into layers (e.g., presentation, business logic, data access) to separate concerns and simplify modifications.
- **Coding Standards and Documentation**: Maintain high code quality with consistent coding standards and thorough documentation to ease understanding and changes.
- **Configuration Management**: Support external configuration methods such as setting files, to allow for changes in behaviour without code modifications.

6.6.4 Trade-offs with Other Quality Attributes

- **Performance**: Abstraction layers and modular designs may introduce overhead, potentially affecting system performance.
- **Complexity**: Designing for high modifiability can increase initial system complexity and development time.
- **Security**: Increased modularity and configurability may introduce security vulnerabilities if not managed properly.
- **Reliability**: Frequent modifications or updates may introduce bugs, potentially impacting system reliability.
- **Cost**: Investing in a highly modifiable architecture may increase initial development costs, especially regarding design. However, these costs can be recovered in the long term, as a modular architecture makes it easier to implement changes or updates.

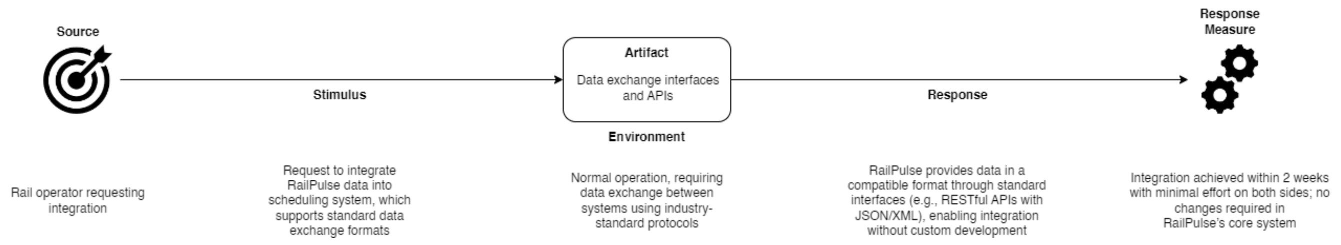
6.7 Integrability

Integrability is essential for the RailPulse system to interact seamlessly with various external systems, such as rail operators' scheduling systems, passenger information systems, and data providers (regional dispatch zones or GPS providers). High integrability ensures that the system can be widely adopted by different stakeholders, facilitating data exchange, enhancing operational efficiency, and promoting interoperability across the rail network. A system that is easy to integrate reduces the effort and cost for stakeholders to adopt and utilise RailPulse services.

We refine integrability into the following sub-attributes:

- **Compliance with standards**
- **Interoperability**

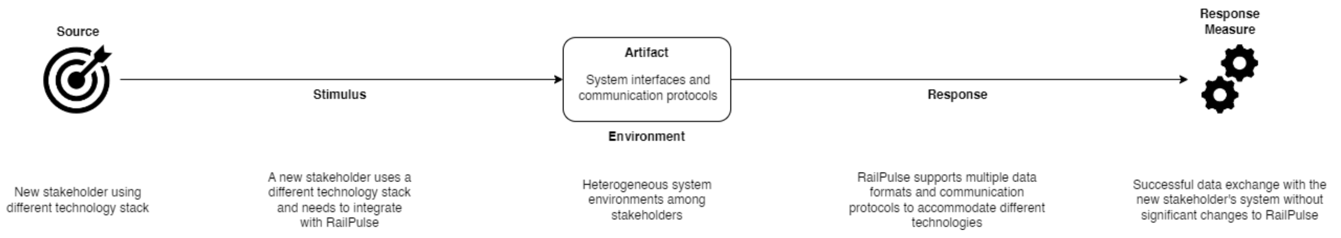
6.7.1 Compliance with Standards - ASR Scenario: Integration with rail operators' scheduling systems using standard protocols


Source
Stimulus
Artifact
Data exchange interfaces and APIs
Environment
Response
Response Measure
Rail operator requesting integration
Request to integrate RailPulse data into scheduling system, which supports standard data exchange formats
Normal operation, requiring data exchange between systems using industry-standard protocols
RailPulse provides data in a compatible format through standard interfaces (e.g., RESTful APIs with JSON/XML), enabling integration without custom development
Integration achieved within 2 weeks with minimal effort on both sides; no changes required in RailPulse's core system

Business Value: High. Compliance with standard protocols ensures efficient integration with rail operators' systems, reducing adoption barriers and facilitating interoperability. Additionally, it is a hard requirement for Railpulse to efficiently interface with the systems of rail operators otherwise, the information it provides is of no use.

Technical Risk: Low. Achieving compliance mainly involves adhering to well-defined industry standards and implementing appropriate interfaces, both of which are common practices. Standard protocols are typically well-documented and supported, simplifying integration and minimizing the risk of technical issues during implementation.

6.7.2 Interoperability - ASR Scenario: Interacting with Diverse Systems and Technologies


Source
Stimulus
Artifact
System interfaces and communication protocols
Environment
Response
Response Measure
New stakeholder using different technology stack
A new stakeholder uses a different technology stack and needs to integrate with RailPulse
Heterogeneous system environments among stakeholders
RailPulse supports multiple data formats and communication protocols to accommodate different technologies
Successful data exchange with the new stakeholder's system without significant changes to RailPulse

Business Value: High. RailPulse is fundamentally an information provider; if it cannot interface effectively with scheduling systems and passenger update apps, its data becomes redundant. Interoperability ensures that its information is usable across a diverse ecosystem of tools, from legacy desktop apps to modern mobile and IoT devices.

Technical Risk: High. Integrating with a diverse ecosystem of external systems, including poorly designed or legacy software, poses significant challenges. RailPulse may need to adapt to constraints outside its control, increasing architectural and development complexity along with the risk of integration failures.

6.7.3 Patterns and Tactics

- **Standard(ised) Protocols**: Employ industry-standard data exchange protocols and data formats to facilitate compatibility.
- **Service-Oriented Architecture (SOA) and Plug-in Architecture**: Design the system as a set of loosely coupled services that can be accessed independently, promoting interoperability. Also, allow integration modules to be added or modified without affecting the core system.
- **API Gateway(s)**: Implement API gateways to manage and expose APIs to external systems securely and efficiently.
- **Data Transformation and Mapping**: Provide data transformation capabilities to accommodate different data formats and structures required by stakeholders.
- **Documentation and Developer Support**: Provide comprehensive API documentation and support to assist stakeholders in integrating with RailPulse.

6.7.4 Trade-offs with Other Quality Attributes

- **Security**: Open interfaces, increased points of integration, and more separate modules may increase the risk of security issues, requiring more thought about security measures.
- **Complexity**: Supporting diverse integration needs may increase system complexity, impacting maintainability.
- **Modifiability**: More separated modules are often beneficial for improving modifiability, However having many separate such modules that communicate via interfaces can introduce constraints to system evolution. If some of these interfaces provide insufficient functionality, it can hinder the project's ability to evolve, by enforcing interface updates for every module update. This limitation may complicate future modifications to the system.
- **Cost**: Additional development and maintenance efforts are required to support integrability features.

6.8 Usability

While Usability is typically a critical quality attribute for systems that directly interface with end-users, in the context of the RailPulse system, it is not a primary concern. RailPulse functions primarily as a backend data provider, supplying real-time data to various external systems such as rail operators' scheduling systems, passenger information systems, and analytics platforms. These external systems are responsible for providing user interfaces and interactions for operational teams and passengers.

In this context, the usability of RailPulse is inherently linked to its **Integrability**. By ensuring seamless integration with external systems, RailPulse indirectly contributes to the overall usability experienced by end-users. The ease with which stakeholders can integrate RailPulse's data into their own systems enhances their ability to develop user-friendly applications.

Therefore, while RailPulse does not directly address usability in terms of user interface design, it supports usability indirectly by:

- **Providing Well-Documented APIs**: Offering comprehensive documentation and developer support to facilitate integration.
- **Adhering to Industry Standards**: Using standard protocols and data formats to simplify integration efforts.
- **Ensuring Data Consistency and Quality**: Delivering accurate, timely, and reliable data that downstream systems can depend on.
- **Supporting Flexible Integration Options**: Allowing data access through various methods (e.g., APIs, data feeds) to accommodate different stakeholder needs.

By focusing on these aspects, RailPulse enhances the usability of its data services, enabling external systems to create intuitive and efficient user experiences. This approach aligns with the system's primary goal of being a reliable and integrable data provider rather than a direct user-facing application.

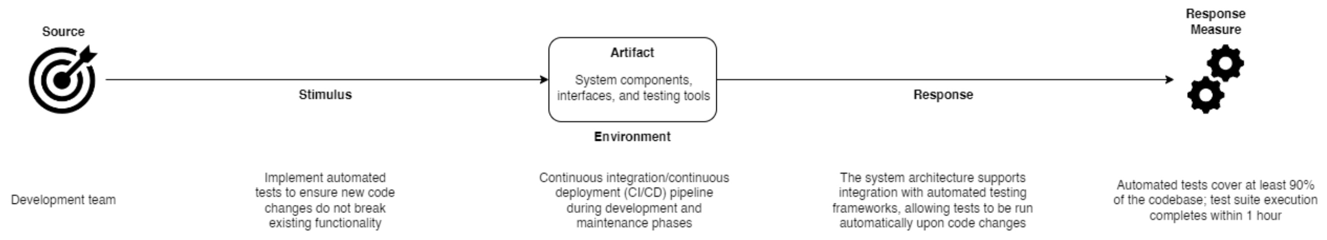
6.9 Testability

As with most software, testability is an important quality aspect, even more so considering the critical nature of operations that RailPulse's system must handle. High testability reduces the time and effort required for testing, facilitates early detection of defects, and improves overall system reliability. By focusing on testability, the development and maintenance teams can validate system functionalities more effectively, leading to higher quality and more robust software that meets stakeholder expectations.

We refine testability into the following sub-attributes:

- **Automated Testing Support**
- **Test Coverage**
- **Reproducibility**

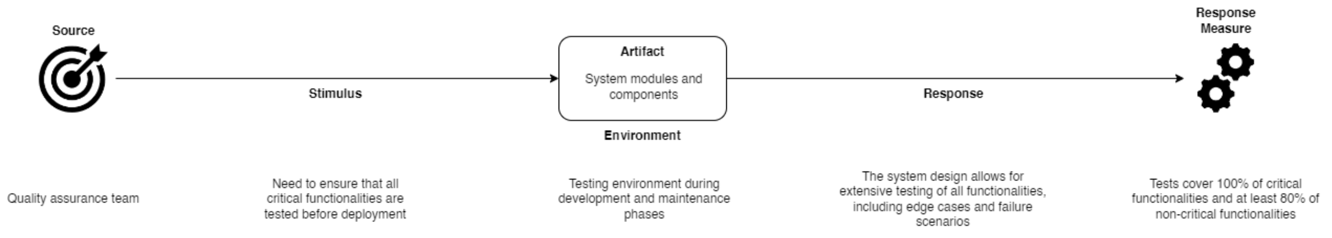
6.9.1 Automated Testing Support - ASR Scenario: Integration with automated testing frameworks


Source
Stimulus
Artifact
System components, interfaces, and testing tools
Environment
Response
Response Measure
Development team
Implement automated tests to ensure new code changes do not break existing functionality
Continuous integration/continuous deployment (CI/CD) pipeline during development and maintenance phases
The system architecture supports integration with automated testing frameworks, allowing tests to be run automatically upon code changes
Automated tests cover at least 90% of the codebase; test suite execution completes within 1 hour

Business Value: Medium. Enhances code quality, reduces manual testing effort, and accelerates development cycles.

Technical Risk: Low. Integrating automated testing frameworks is a standard practice supported by mature tools and technologies. With proper initial setup, the process is well-documented and straightforward, minimising the likelihood of major implementation challenges.

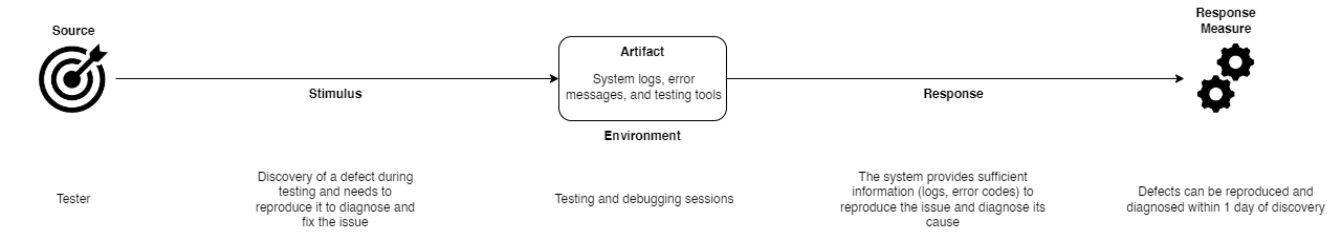
6.9.2 Test Coverage - ASR Scenario: Comprehensive testing of system functionalities


Source
Stimulus
Artifact
System modules and components
Environment
Response
Response Measure
Quality assurance team
Need to ensure that all critical functionalities are tested before deployment
Testing environment during development and maintenance phases
The system design allows for extensive testing of all functionalities, including edge cases and failure scenarios
Tests cover 100% of critical functionalities and at least 80% of non-critical functionalities

Business Value: High. Comprehensive testing ensures that critical functionalities are thoroughly validated, preventing the risk of bugs which might have severe safety implications.

Technical Risk: Low. High test coverage can be achieved using established testing frameworks and tools, which provide extensive support for unit, integration, and system testing. With a modular architecture and clear testing practices, implementing comprehensive test coverage becomes a predictable and manageable process.

6.9.3 Reproducibility - ASR Scenario: Ability to reproduce and diagnose issues found during testing


Source
Stimulus
Artifact
System logs, error messages, and testing tools
Environment
Response
Response Measure
Tester
Discovery of a defect during testing and needs to reproduce it to diagnose and fix the issue
Testing and debugging sessions
The system provides sufficient information (logs, error codes) to reproduce the issue and diagnose its cause
Defects can be reproduced and diagnosed within 1 day of discovery

Business Value: Low to Medium. The ability to reliably reproduce and diagnose issues ensures efficient debugging and reduces resolution times. Still, it is not of critical importance to the overall system functionality, as long as all the bugs found during testing are addressed before deployment.

Technical Risk: Low. Reproducibility is supported by well-established practices such as logging, structured error handling, and the use of debugging tools. These techniques are widely documented and straightforward to implement in modern software systems.

6.9.4 Patterns and Tactics

- **Modular Design**: Structure the system into independent, testable units to simplify testing.
- **Automated Testing Tools**: Integrate with automated testing frameworks (e.g., JUnit, pytest) to support unit, integration, and system tests.
- **Continuous Integration (CI)**: Use CI pipelines to run automated tests upon code commits, ensuring early detection of defects.
- **Detailed Logging**: Implement comprehensive logging to capture system behavior and facilitate issue diagnosis.
- **Test Data Management**: Use standardised test data sets to ensure consistency and reproducibility in tests.
- **Mocking and Stubbing**: Use mocks and stubs to simulate components or services that are not under test.

6.9.5 Trade-offs with Other Quality Attributes

- **Development Time and Cost**: Achieving high testability may increase initial development effort and costs.

6.10 Maintainability

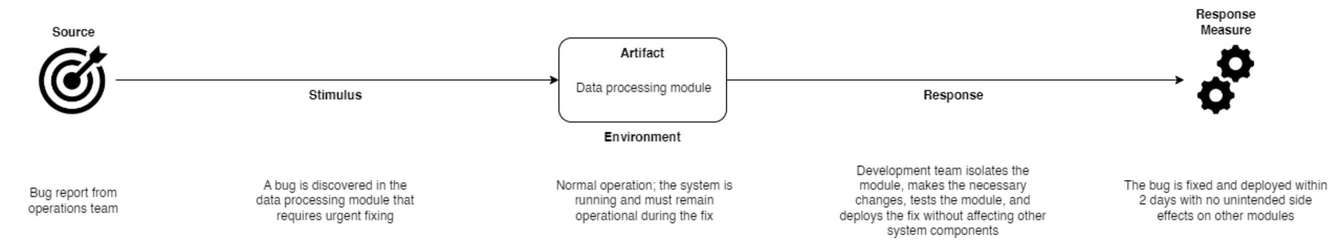
Finally, even though not critical for the project to meet its main requirements, maintainability is nevertheless a notable quality aspect, promoting the system's longevity (i.e., through ease of maintenance, update, and improvement). High maintainability reduces the time and cost associated with fixing defects, implementing new features, and adapting to changing requirements. By focusing on maintainability, the system can remain reliable, secure, and performant, while extending its useful lifespan and providing ongoing value to stakeholders.

We refine maintainability into the following sub-attributes:

- **Modularity**

- **Code Quality**
- **Documentation**

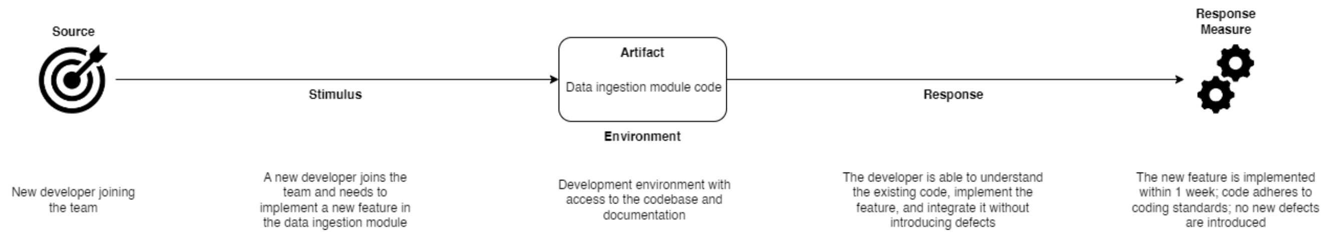
6.10.1 Modularity - ASR Scenario: Efficient isolation and modification of system components


Source
Stimulus
Artifact
Data processing module
Environment
Response
Response Measure
Bug report from operations team
A bug is discovered in the data processing module that requires urgent fixing
Normal operation; the system is running and must remain operational during the fix
Development team isolates the module, makes the necessary changes, tests the module, and deploys the fix without affecting other system components
The bug is fixed and deployed within 2 days with no unintended side effects on other modules

Business Value: Medium. Modularity simplifies isolating and modifying components, reducing downtime and maintenance effort. While it enhances system flexibility, its benefits are primarily visible during development and maintenance rather than directly impacting end-users.

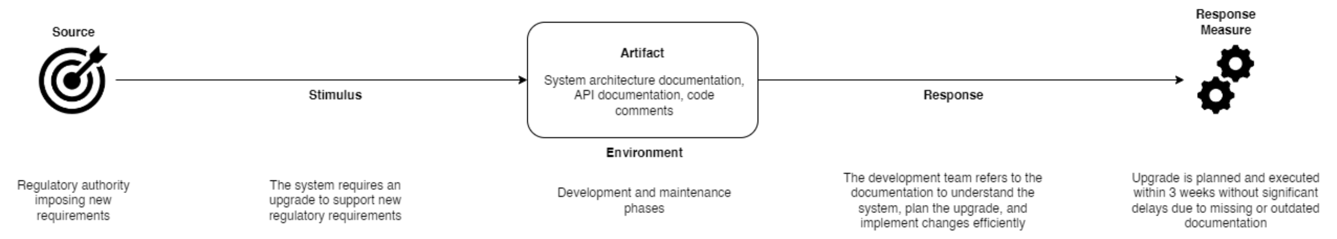
Technical Risk: Medium. Designing a modular architecture requires thorough planning and adherence to principles like separation of concerns, which can increase initial development complexity. However, it is a common design pattern familiar to the majority of software developers.

6.10.2 Code Quality - ASR Scenario: Ease of understanding and modifying the codebase


Source
Stimulus
Artifact
Data ingestion module code
Environment
Response
Response Measure
New developer joining the team
A new developer joins the team and needs to implement a new feature in the data ingestion module
Development environment with access to the codebase and documentation
The developer is able to understand the existing code, implement the feature, and integrate it without introducing defects
The new feature is implemented within 1 week; code adheres to coding standards; no new defects are introduced

Business Value: Low to Medium. Accelerates development, reduces errors, and facilitates onboarding of new team members. Although, the majority of benefits are not visible to external stakeholders. **Technical Risk:** Low. Requires adherence to coding standards and code review processes which are implicit requirements to most professional software development teams.

6.10.3 Documentation - ASR Scenario: Availability of up-to-date and comprehensive system documentation


Source
Stimulus
Artifact
System architecture documentation, API documentation, code comments
Environment
Response
Response Measure
Regulatory authority imposing new requirements
The system requires an upgrade to support new regulatory requirements
Development and maintenance phases
The development team refers to the documentation to understand the system, plan the upgrade, and implement changes efficiently
Upgrade is planned and executed within 3 weeks without significant delays due to missing or outdated documentation

Business Value: Medium. Facilitates efficient maintenance and reduces the risk of errors during bug fixes or updates. **Technical Risk:** Low. Requires ongoing effort to maintain documentation but it is nevertheless an implicit requirement from good quality software development.

6.10.4 Patterns and Tactics

- **Modular Architecture**: Design the system with clear separation of concerns, enabling independent development and modification of components.
- **Coding Standards**: Establish and enforce coding standards to ensure consistency and readability across the codebase.
- **Code Reviews**: Implement code review processes to maintain code quality and share knowledge among team members.

- **Documentation Practices**: Maintain comprehensive and up-to-date documentation, including system architecture diagrams, API references, and inline code comments.
- **Refactoring**: Regularly refactor code to improve structure, reduce complexity, and address technical debt.

6.10.5 Trade-offs with Other Quality Attributes

- **Performance**: Highly modular systems may introduce overhead due to abstraction layers, potentially affecting performance. Even though such side-effects can arguably be minimal, they can still pose an important threat to systems which deal with real-time, safety-critical data (like the one offered by Railpulse).
- **Development Time and Cost**: Investing in maintainability (e.g., documentation, code reviews) may increase initial development time and costs.

6.11 Connecting Quality Attributes and Stakeholders

For the key stakeholders, we have determined which quality attributes they are interested in, we classify them as either high, medium or low interest.

Availability	Rail Operators (High), Rail Infrastructure Owners (High), Operational Teams (High), Railway Maintenance Team (High), Data Providers (High), EU (High)
Performance	Operational Teams (High), EU (High)
Security	Security Team (High), Regulatory Bodies (Medium), Data Providers (Medium), EU (Medium)
Safety	Railway Maintenance Team (High)
Deployability	Software DevOps Team (Medium)
Modifiability	Software DevOps Team (High), EU (High)
Integrability	Data Providers (High), Rail operators (Medium)
Usability	Operational Teams (High), Rail Operators (High), Freight Customers (High), Passengers (High), Rail Infrastructure Owners (Medium), Railway Maintenance Team (Medium)
Testability	Software DevOps Team (High)
Maintainability	Software DevOps Team (High), EU (High)

6.12 QA Prioritisation

By identifying the key quality attributes and associated scenarios, patterns, and trade-offs, we have established a foundation for the architectural design of the RailPulse system. Nevertheless, an exhaustive architecture is naturally unattainable due to implicit contextual constraints as well as limitations related to development time and resources. Thus, a prioritisation of quality attributes is imperative in order to structure and guide the design process, ensuring that the final system meets the most significant stakeholder concerns.

To this end, the utility tree constructed from each (refined) QA, along with the ”business value” and ”technical risk” criteria associated with each scenario, served as the baseline for prioritising the quality attributes most relevant to the system, along with the ones which should be of little concern. Still, before these are explored, it is important to mention the caveat that by ”low priority” we do not mean QA which should have little to no effort put into achieving them. Instead, we imagine the priority scale to start from the point in which the QA is met in accordance with all regulatory standards and industry practices. Consequently, the lowest-priority attributes receive just enough effort to meet industry-enforced minimum levels of correctness, as no further investment is deemed necessary. In contrast, higher-priority attributes exceed these baseline standards, requiring additional time and resources to implement solutions that go ”above and beyond”, providing enhanced functionality essential to the system’s success.

This being said, by looking at the utility tree entire diagram available in Appendix C) one can clearly see four main quality attributes which should be prioritised in the RailPulse system’s architecture, along with two QAs of less relative importance. Composing the former category: safety, availability, performance, and integrability represent aspects critical to the proper functioning of certain ”high-stakes” parts of the system. Since the information provided by RailPulse is essential for making safe and timely train scheduling decisions, the data provided by the system should be accurate (Safety), accessible (Availability), fresh (Performance), and able to be used by the various interfaces which require and make use of it (Integrability). The first three QAs ensure the relevance of the information, while the fourth guarantees it can be actually utilised by the systems requiring it.

As for the low-priority attributes, we have identified Deployability and Maintainability to be of less relative importance. Again, it is important to stress the fact that this classification does not mean a neglect of the two system aspects, but rather indicates that the industry-standard level of functionality is deemed sufficient, with no need to allocate additional development resources to enhance these aspects beyond regulatory and practical requirements. Considering that deployment would be a relatively rare operation, standard installation operations should suffice, as possible errors won't pose a big/often-enough inconvenience. Similarly, standard documentation and code quality practices should be enough, as too much effort invested into such endeavours could have diminishing returns [8].

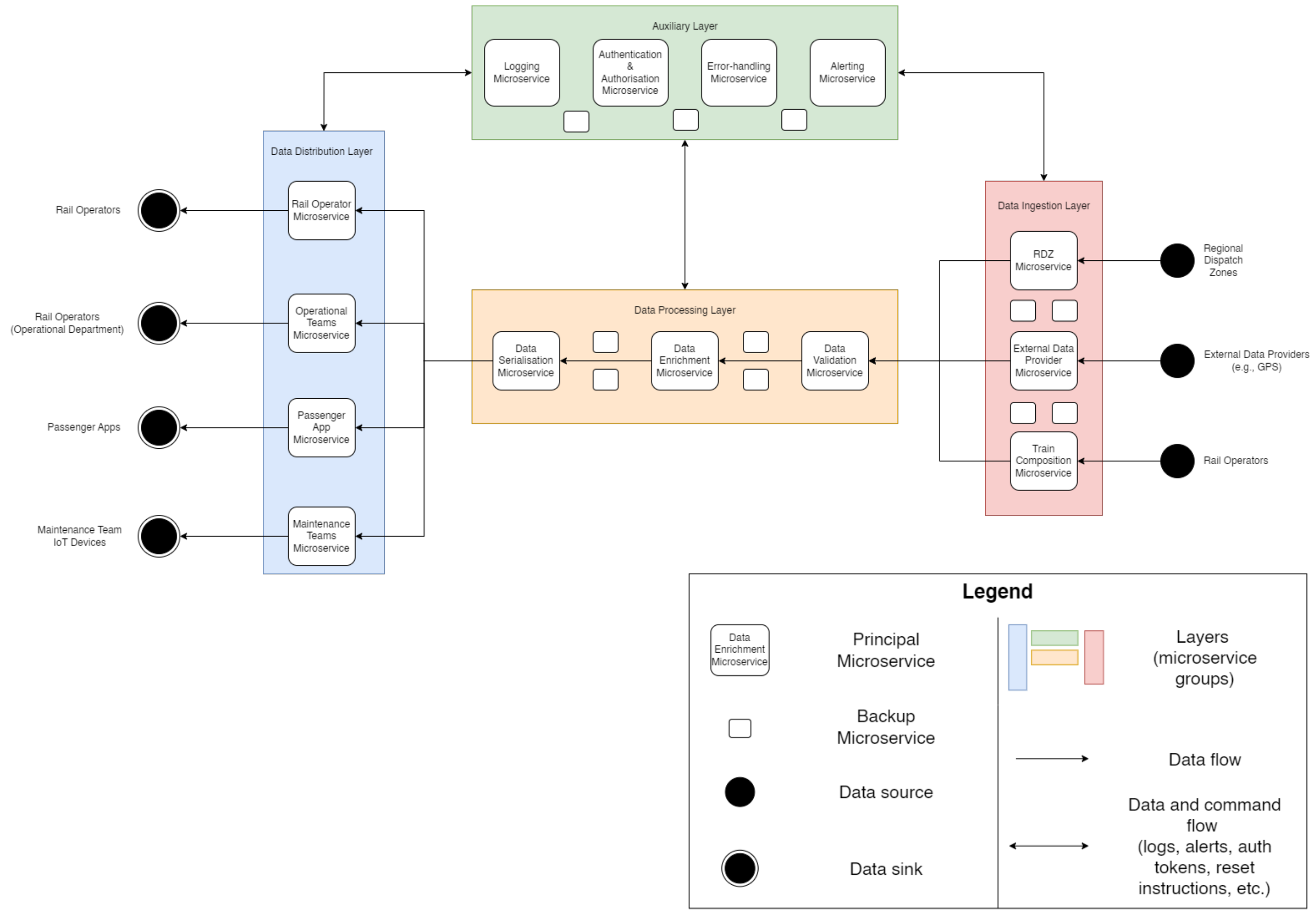
6.13 Motivation for Response measure thresholds

A last important note regarding the description of the chosen quality attributes is the rationale behind the response measure thresholds. These were determined based on the authors' professional experience and industry knowledge, reflecting what would typically be expected for a system of RailPulse's scope and criticality. Additionally, the values also reflect the prioritisation of QA in accordance with stakeholder concerns, as presented in Section 6.12. For example, the high importance of reliability (availability) justified the strict criterion of 99.99% system uptime over a year, which is relevant for such a safety-critical system. In contrast, if availability had been of lower priority, a more lenient threshold, such as 95-97%, might have sufficed.

Nevertheless, while these estimations align with industry standards and practices for similar systems, the hypothetical nature of this project introduces certain limitations. The absence of direct stakeholder or domain expert input means that the thresholds cannot be fully validated within the scope of this paper. In a real-world context, such measures would be extensively reviewed and confirmed through stakeholder consultations, ensuring they accurately reflect operational needs while also managing expectations.

7 HIGH-LEVEL ARCHITECTURE

7.1 Logical View



Auxiliary Layer
Logging Microservice
Authentication & Authorisation Microservice
Error-handling Microservice
Alerting Microservice
Data Distribution Layer
Rail Operators
Rail Operator Microservice
Rail Operators (Operational Department)
Operational Teams Microservice
Passenger Apps
Passenger App Microservice
Maintenance Team IoT Devices
Maintenance Teams Microservice
Data Processing Layer
Data Serialisation Microservice
Data Enrichment Microservice
Data Validation Microservice
Data Ingestion Layer
RDZ Microservice
Regional Dispatch Zones
External Data Provider Microservice
External Data Providers (e.g., GPS)
Train Composition Microservice
Rail Operators
Legend
Data Enrichment Microservice
Principal Microservice
Backup Microservice
Data source
Data sink
Layers (microservice groups)
Data flow
Data and command flow (logs, alerts, auth tokens, reset instructions, etc.)

Figure 5. Logical view - Container diagram [6] of the microservice-based architecture of RailPulse

Even though this system architecture mainly follows the C4 model [6], we have taken inspiration from the 4+1 Architectural View Model [16] in defining the highest-level view of RailPulse.

In accordance with the C4 model, the Logical View (inspired from 4+1) in Figure 5 represents the container level, building on the system context diagram detailed in Figure 1. This view outlines the primary layers and their respective containers within the RailPulse system, showcasing their high-level purpose, responsibilities and interactions. It acts as a bridge between the broad overview provided by the system context and the more detailed figures, offering stakeholders, auditors, regulators and developers a clear but concise understanding of the system's architecture.

The purpose of this view extends beyond visualising the system's structure; it provides a foundation for achieving the system's functional and quality objectives. More specifically, this view encapsulates how architectural patterns are applied to address specific challenges posed by RailPulse's safety-critical and data-intensive nature. By embedding the patterns defined in Section 7.2 into the logical view it ensures the system's robustness and adaptability, fostering a design capable of adhering to the currently most important concerns while remaining flexible for future evolution. In the following section, we provide an in-depth description of the Logical View, through the lens of the architectural patterns used for creating this view.

7.2 Architectural Patterns

This section presents the architectural patterns selected for the RailPulse system Logical View. It explains their relevance to the system's requirements, outlines their contributions to achieving key quality attributes, and highlights how they address challenges unique to the RailPulse context. By grounding the system's architecture in established patterns, we ensure both robust functionality and the ability to adapt to future demands.

7.2.1 Monolith vs. Microservices

Considering that RailPulse has to interact with multiple data sources and sinks, all with different interactions methods that are outside of our control, there is an inherent need for separation of concerns, a requirement for implementing different interaction methods, and the need to be able to easily add new data sources/sinks or extend or alter already existing services.

The main choice here boils down to whether to adopt a monolithic architecture or to split the architecture up into microservices. A monolithic architecture is the ”traditional” software development model where each concern in the system is inherently connected to each other, due to sharing the same process context and due to data exchange mechanisms in the system [1]. Microservices, on the other hand, aim to reduce the interdependency of components by physically splitting them up into different system contexts, such as putting them in separate Virtual Machines or containers.

While a monolith would be faster to develop for a smaller team and would be more performant due to reduced communication costs, we believe a microservice-based design is the best choice here. This is because we prioritise the concerns System Reliability, Fault Tolerance, Minimal Downtime, Modularity, and Scalability over concerns such as Low Resource Utilisation (see Table 3), which translates to the prioritisation of Quality Attributes such as Availability and Modifiability. A monolith would be worse in terms of Availability, because special care needs to be taken in order to ensure the entire system does not go down when one part critically fails, and by its nature, the entire system must be hosted on one machine/container, which further reduces reliability. On the other hand, singular microservices can easily be restarted, and a backup system can be made to ensure that if these services fail, there is always another that can take the load. While this does introduce extra costs, and care should be taken to not place backup services in similar regions to prevent outages if a cloud region goes down, the cost is made up by the ability to split up and backup different parts of the software.

Furthermore, a microservice architecture is inherently scalable and modular, allowing for allocating more resources to a particular service such as data processing, while a monolith would not have this flexibility in resource allocation. This is because it is easier to interconnect different concerns in a monolith, while these would be separate services entirely in a microservice architecture and can be scaled and expanded independently.

Now, while microservices have the possibility to be more performant [3], they do sacrifice latency by their very nature of having to talk to each other over the internet[4], and are inherently more complex to implement. However, latency in this case is of lesser importance, as we consider ’real-time’ to be within a couple of seconds.

For the main system tasks, microservices can be divided into distinct layers, each addressing a specific aspect of the RailPulse architecture. The **data ingestion layer** includes services for each data source, such as regional dispatch zones, GPS providers, and train composition systems - all having distinct requirements and utilising different data formats. This would allow for dedicated interfacing with each information provider and providing a central place for translating that data into a common format used by RailPulse. Subsequently, the formatted data is passed down to the **data processing layer**, which contains microservices for tasks such as data validation, enrichment (e.g., merging GPS data with dispatch information), and serialisation (e.g., transforming data into a format suitable for real-time train tracking). By separating these features both conceptually (via the architecture) and implementation-wise (via their own dedicated service), the final product ultimately becomes easier to maintain and update. Next, the **data distribution layer** mimics the structure of the data ingestion one, hosting dedicated services for each specific data sink, such as rail operator scheduling systems, passenger information apps, freight logistics platforms, and even IoT devices used by maintenance teams. This separation ensures that the system can offer customized data, formats, protocols, and access patterns to each recipient, enhancing interoperability and security.

The **auxiliary layer**, a set of dedicated microservices addresses cross-cutting concerns that are essential to the system’s operation. This set of services serves as a wrapper for the entire system, by ensuring things such as secure access, monitoring, and logging. Some authentication service can implement authentication/authorisation mechanisms, a logging service can be implemented to centralise logs in order to facilitate troubleshooting or help with regulatory compliance, error-handling microservices can provide consistent management of faults, from detecting errors to applying fail-safe responses and escalating unresolved issues, and a configuration management microservice can handle system-wide parameter changes (e.g., data retention policies or access permissions), allowing administrators to update settings dynamically without requiring system restarts or code modifications.

By dividing core and auxiliary tasks into distinct microservices, the RailPulse system can achieve a highly modular, scalable, and maintainable design. Nevertheless, microservices can introduce complexities such as higher operational overhead and a potentially more work-intensive and complex setup, but we find these to be of less significance, as the additional overhead can be mitigated through efficient communication protocols and more resource allocation, while complexity is not a prioritised quality attribute.

[4]Even local TCP or UDP communication is logically quite a lot slower than copying or passing around a region of memory

7.2.2 Serverless vs. server-based architecture

Coupling well with the microservice-based architecture is a serverless architecture, which enhances scalability and cost efficiency by dynamically allocating resources to match fluctuating workloads. Train traffic - the main object of interest for RailPulse, is characterised by peak hours of intense activity and off-peak periods of relative inactivity, which means that the system could benefit significantly from this dynamic resource allocation. A serverless infrastructure ensures sufficient processing power during high-demand periods while reducing operational costs during quieter times, such as nighttime for train activity. This capability is particularly advantageous for RailPulse, where system responsiveness must remain uncompromised regardless of load variability.

Another compelling argument for serverless architecture is its rapid deployment capabilities and reduced time-to-market. By abstracting away the underlying infrastructure management, serverless solutions allow developers to focus exclusively on implementing business logic. This benefit aligns with RailPulse's need to accommodate diverse stakeholders and evolving requirements, enabling the system to integrate new data sources or sinks without significant delays. Moreover, the stateless nature of serverless functions (a part of serverless architecture), enhances modularity and maintainability, as each function operates independently and can be updated or replaced without affecting the broader system.

However, serverless architecture is not without challenges, particularly in the context of performance, availability, and security. Reliance on cloud providers can increase latency in the case of cold starts [30] or data transfer overheads, although such drawbacks can be mitigated through architectural optimizations, such as keeping frequently used functions "warm" and using efficient data serialization protocols. When it comes to availability, the issue is more pronounced in a safety-critical system such as RailPulse, as it becomes entirely reliant on a third-party provider. Downtime or service disruptions on the provider's end could lead to a complete halt in the system's functionality, posing significant risks to operations. Security concerns also arise, as sensitive data flowing through cloud-managed infrastructure requires robust encryption and access control measures to protect against breaches. These risks are acknowledged, and further details on mitigation strategies are discussed in Section 10.

By using a serverless architecture we deal with the concerns mentioned in Section 5.13 and Section 5.14. Scaling is much easier, since you can buy more resources from your provider when needed. Additionally, you do not always have to pay full price when you do not use all resources, which can result in cost savings, during for example the night when trains do not run. The serverless architecture also helps us improve the QA Maintainability, as discussed in Section 6.10. When using such architecture you do not have to do any server maintenance yourself, this reduces the amount of money and time spent on maintenance.

We have also considered the alternative to a serverless architecture: hosting it ourselves. While this offers some advantages, like control over the hardware, we believe the flexibility, scalability and cost savings a serverless system provider outweigh the small advantages.

7.2.3 Event-Driven vs. Non-Event-Driven Architecture

Another possible architectural pattern to choose an event-driven one, which is more suitable for handling asynchronous communication and real-time data processing. This is particularly applicable to RailPulse, which has to process real-time train location updates and must respond dynamically to system events, such as delays or changes in train direction. For instance, a delay notification from a dispatch zone can trigger an event consumed by the data distribution microservice, updating passenger apps and scheduling systems in real time. Similarly, maintenance alerts from IoT devices could initiate event-driven workflows to notify operational teams promptly.

The alternative to this would be a non-event-driven architecture, where events are 'polled', i.e., information has to be actively queried. This model is more suitable where information is needed on-demand, and where the delays between the data being available and the data being used/processed/retrieved are not of priority. This model introduces delays and extra overhead, especially internally within the microservices, as each microservice would have to poll all other dependent microservices continuously for data, which would cause large unneeded network loads. Furthermore, utilising an event-driven architecture allows for the decoupling between components, as event producers (e.g., data ingestion services) are not directly tied to event consumers (e.g., distribution services for scheduling systems or passenger apps). This enhances system scalability and maintainability, as new components can be added or updated without impacting the rest of the system. For example, RailPulse could introduce new event consumers to process maintenance alerts without modifying the existing train tracking infrastructure. Additionally, this pattern supports efficient resource usage separate from network load, as components are only activated when relevant events occur, which aligns well with the system's need to manage varying workloads (thus synergising with a serverless architecture as well).

However, event-driven architecture introduces challenges, particularly regarding event prioritisation. In a safety-critical system such as RailPulse, there is a risk of high-priority events (e.g. real-time alerts for train delays) being stuck in a backlog due to less important but high-volume events (e.g. routine data synchronisation). Such scenarios could hinder the timely processing of critical information, impacting operational decisions and overall safety. To mitigate this risk, RailPulse's architecture can implement priority-based queues and advanced message routing strategies, ensuring that critical events are processed with precedence. For a deeper exploration of these concerns and their mitigation, the reader is directed to Section 10.

With this event-driven architecture, we deal with the concern in Section 5.11. The event-driven architectures also help with sending the right data to the different microservices, this then also improves the scalability, dealing with the concern in Section 5.13. The event-driven method also improves on the quality attribute Performance discussed in Section 6.2.

7.3 Data Flow View

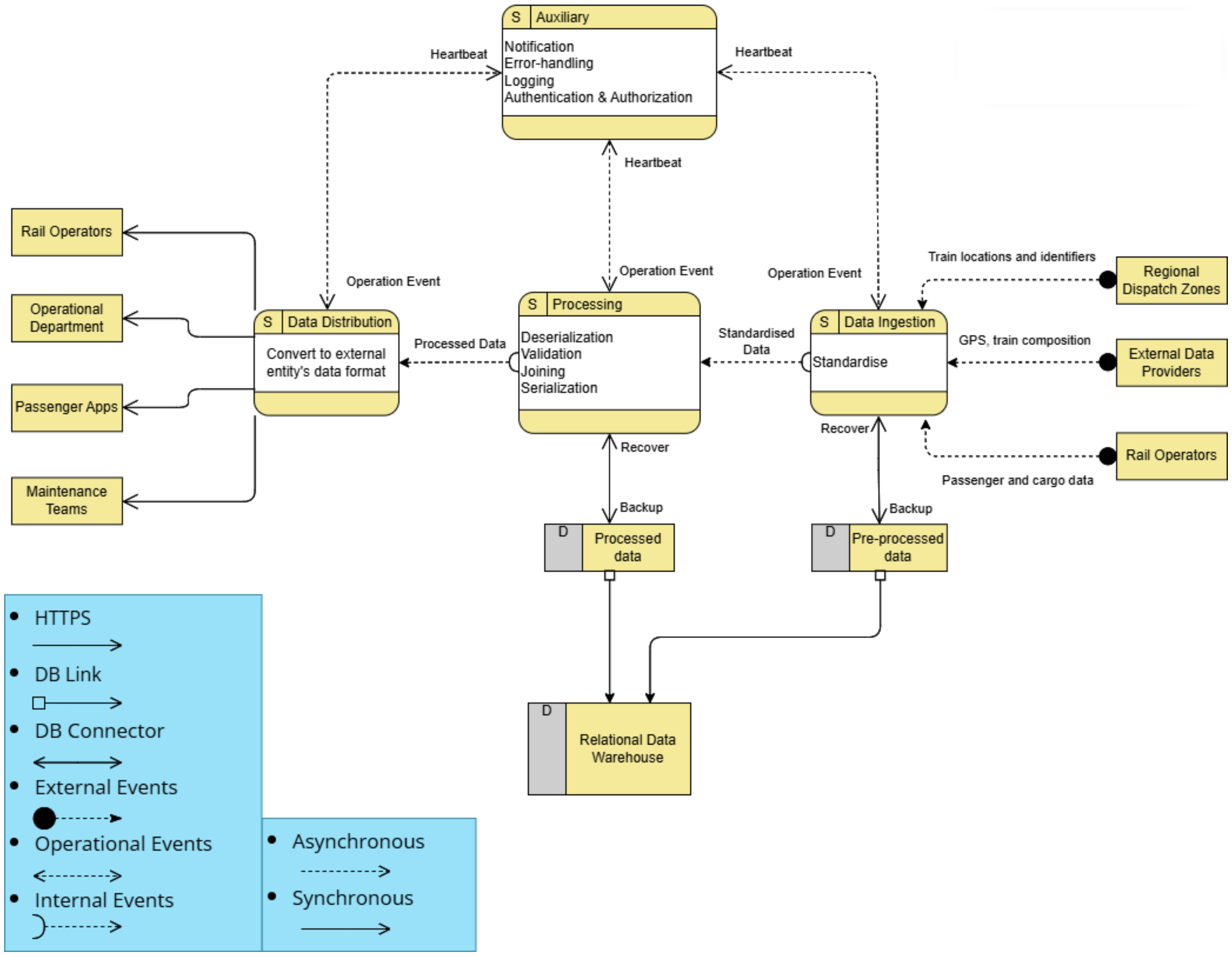


Figure 6. Data Flow in RailPulse

Motivation For many stakeholders, the way in which data flows through the system is an important consideration. The directly-involved actors that hook their systems to the data distribution layer (Figure 5), such as operational teams, maintenance teams and rail operators, are interested in the interfaces and data exchange formats that the RailPulse distribution layer provides (Section 5.7). Moreover, data flow also plays a major role in the efficiency, availability and performance of the system, therefore, it also represents an object of interest for stakeholders that are not directly linked to the system, such as passengers who would like quick response times for their needs (Sections 5.5 and 5.11). Last but not least, the data flow view could also be an object of interest towards regulatory bodies, and even the EU, in terms of data security, compatibility with existing railway communication systems such as GSM-R, and the use of standard interfaces for standardisation purposes (Sections 5.8, 5.13 and 5.16).

Explanation In Figure 6, an overview of the data flow through the RailPulse system is presented at a C4-container level [6]. Starting from right to left, the diagram describes the data flow from the data ingestion layer, all the way through the data distribution layer. According to the event-driven architecture paradigm, all incoming data from the external actors in the environment such as data providers and regional dispatch zones will be wrapped in events, sent asynchronously using an event bus. The data ingestion layer then is tasked to observe incoming events on the bus, consume them according to their priority levels, standardise the data to the RailPulse standardised data exchange format, and then asynchronously transmit the data towards the processing layer. The internal events are transmitted via another event bus, which is restricted to internal (i.e. in between internal layers) communication. Before transmitting its data, the data ingestion layer also backs up all of its output in a key-store, to facilitate quick recovery in case of system failure. The data processing layer deserialises all the incoming events into objects, validates them according to specified validation rules, joins the objects into a standardised decorator object, and transmits the data to the next layer via the internal event bus. The same recovery and backup via a local key-value store is used for the processing layer too. The last layer in the responsibility chain converts all the received events to a common data exchange format which is agreed upon with each data-dependant stakeholder. This last layer is composed of serverless functions that run-on-demand whenever the client needs data. Linked to all layers is an auxiliary layer, which is tasked with providing authorization tokens, logging errors and operations that happen in all the data layers, and error handling by constantly monitoring the state of the data services. If one data microservice is down, the auxiliary layer will trigger a fail safe to restart the faulty microservice. The latest data will then be fetched from the local key-value store. The communication between the auxiliary layer and the data layers is done in two ways. Firstly, the simple messages used to check if a data layer is reachable is done using a heartbeat communication protocol via TCP. With this protocol, important operational messages can be transmitted which can be used for load balancing, important logs, or other high priority low-level messages. The other messages used for authentication, authorization or notifications are published on the operational event bus.

7.4 Deployment View

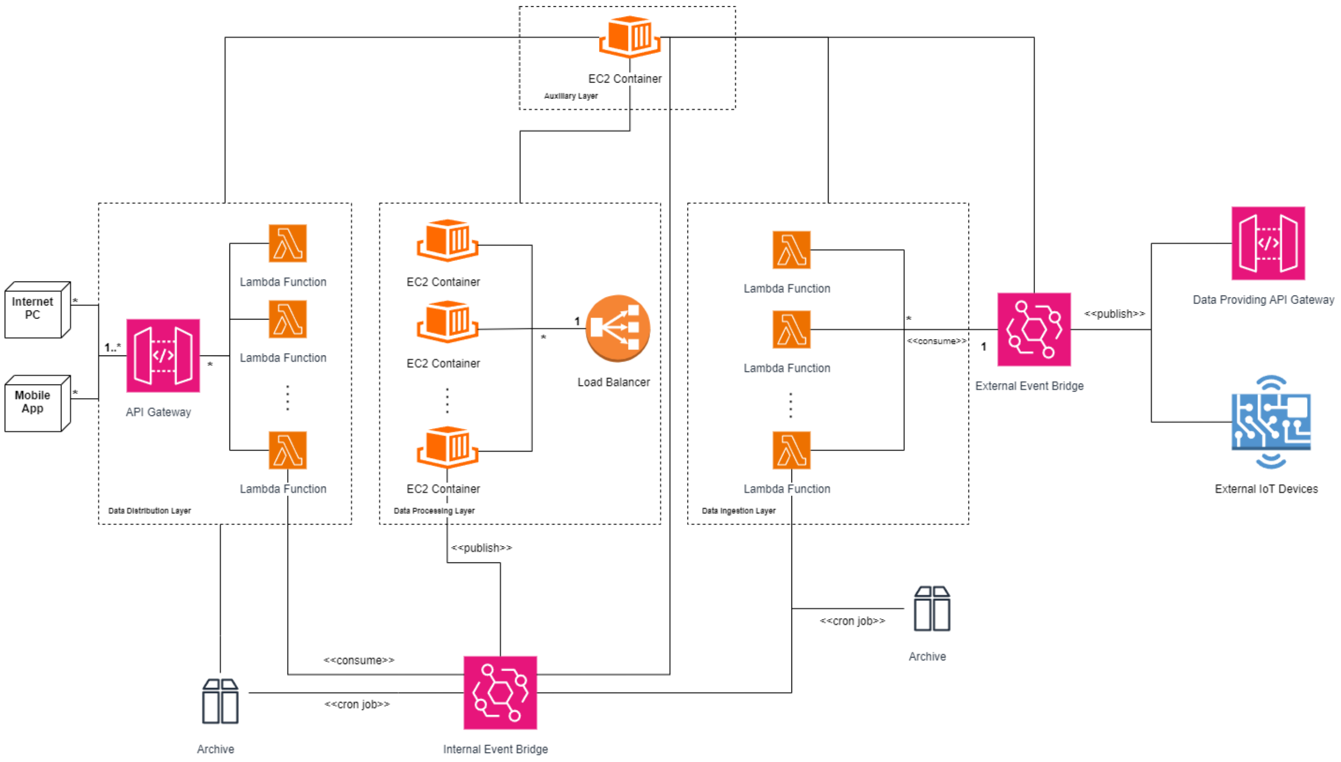


Figure 7. Example Deployment of a RailPulse instance using Amazon's AWS cloud infrastructure.

Motivation According to our stakeholder's concerns, which are presented in Section 5, reliability, fault tolerance and minimal system downtime rank highly on their list of expectations for the RailPulse project. Ensuring that a system of such scale such as RailPulse operates in the margins proposed in Section 6 is a difficult task, especially on self-hosted hardware which is not as scalable as cloud-provided solutions. Taking into consideration the complexity in the deployment process of RailPulse, the deployment view presents itself as an invaluable piece of documentation. Multiple stakeholders, such as software DevOps teams and security teams are interested in such a deployment view for multiple reasons, including technical support in the development, deployment and maintenance phases of the software. Moreover, from a business perspective, RailPulse Inc. might be interested in such a view to analyse and compare similar deployment services offered by multiple cloud providers, and the prices of each provider respectively.

Explanation In Figure 7, an overview of the deployment of the RailPulse system is presented, using Amazon's AWS [29] as an example cloud provider. An important consideration is RailPulse's deployment is not tied to AWS's platform, and it can be ported to any other cloud provider given the need, by using similar services. Starting from right to left, the figure describes the main entry point for data in the system: the External Event Bridge [27], a scalable and robust event queue that enables data producers such as external IoT devices or data provider systems to publish data in the form of events. On the consumer side of the event bridge, multiple Lambda functions [28], more specifically, as many as the system needs based on its load, consume events to standardise their payloads to the RailPulse standardised data exchanged format. The data ingestion layer then publishes events on another event queue, the Internal Event Bridge, which is used only for intra-service communication. Each published message is stored in an Archive for a pre-established amount of time to allow for event replay, in case of data loss, system failure or other errors. The data processing layer consists of multiple EC2 containers [26], which scale up and down with load using a load balancer. The choice of EC2 containers in the processing layer is justifiable: Lambdas are optimised for tasks which last at most 15 minutes, whereas EC2 containers are optimal for long-term operations such as validation, enrichment and serialisation of multiple Gigabytes of data a day. The data distribution layer, similarly to the data ingestion layer, is composed of Lambdas, which are more optimised for high-throughput, short-term loads. The output of the data distribution layer is transmitted through an API Gateway, hosted in the cloud, to all the clients that request it. At last, the auxiliary layer is composed of a single EC2 container which hosts all the auxiliary services. In case of high loads, another container can be started to evenly distribute the load.

Migration between Cloud Providers Throughout our architecture document, we used Amazon AWS specific terms for consistency, but the reality is that RailPulse's system is not tied to any particular cloud provider, and that is a consequence of our choice of simple individual components. Serverless functions, containers and even load balancers and event bridges represent standard components present in multiple providers such as Microsoft Azure, Amazon AWS, Google Cloud or even Alibaba Cloud, albeit, under different naming schemes. However, regardless of which cloud provider is chosen, there could be a potential risk of having to migrate to another one due to any external factors such as a service's end of life. Each cloud provider offers their own APIs for calling serverless functions, or to send messages through their proprietary event bridges, therefore it comes as a given that a migration will have to also take into account changes in all areas of the code. To mitigate the impact of such heavy refactoring, our architecture focuses on separating the business logic from the proprietary APIs used by each provider, so that a migration can be done by just changing a few APIs in specialized modules.

8 COMPONENT SPECIFIC VIEWS

8.1 Data Ingestion Layer

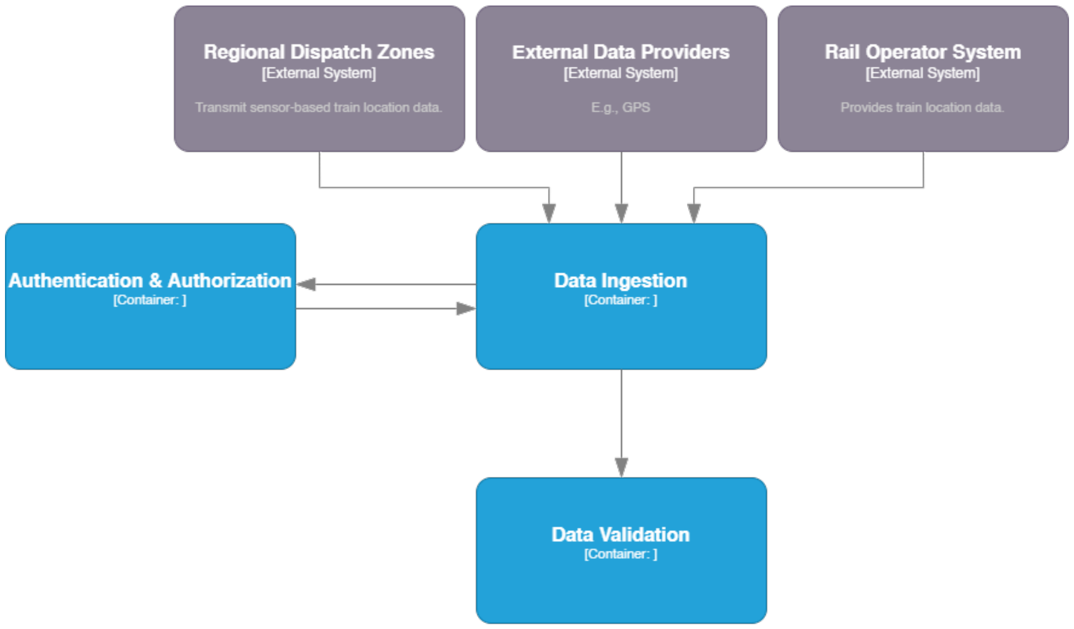


Figure 8. Data Ingestion Layer for RailPulse (Container Diagram)

The data ingestion layer is the first step in the pipeline, where the data is ingested into the system. Figure 8 shows the relevant containers for the Ingestion Layer. In Figure 9 you can see we use 3 micro-services for the Data Ingestion Layer. Since these are very similar and only have some small differences in the data they accept, we will model them once. The Data Ingestion container is the main container for this layer, it receives the data, does some security checks, like checking proper authentication and then passes it on to the Data Validation container in the Data Processing Layer. In order to discuss the Data Ingestion container in more detail we have made a component diagram from this, which can be seen in Figure 9. The container is built out of 3 components, starting with the Data Providers API, this API handles the incoming data from the 3 stakeholders that provide our data. The API component interacts with the components in the Authentication & Authorization container, to ensure that the correct entities are ingesting the data. Before the data can be processed we first check it thoroughly using a security filter, which filters all malicious requests. Afterwards, we can convert the data in the Data Converter components. Since all data that arrives in our system can come from any of the 3 stakeholders (Regional Dispatch Zones, External Data Providers, Rail Operators) that ingest data, there could slight differences in the schemas of incoming data, for example, the regional dispatch zones do not provide the current real-time location of the train but more a ”seen here 1 min ago” format. We process and standardise the data to a common schema, then the data which at last, is forwarded towards the data processing layer, the next layer in the processing chain.

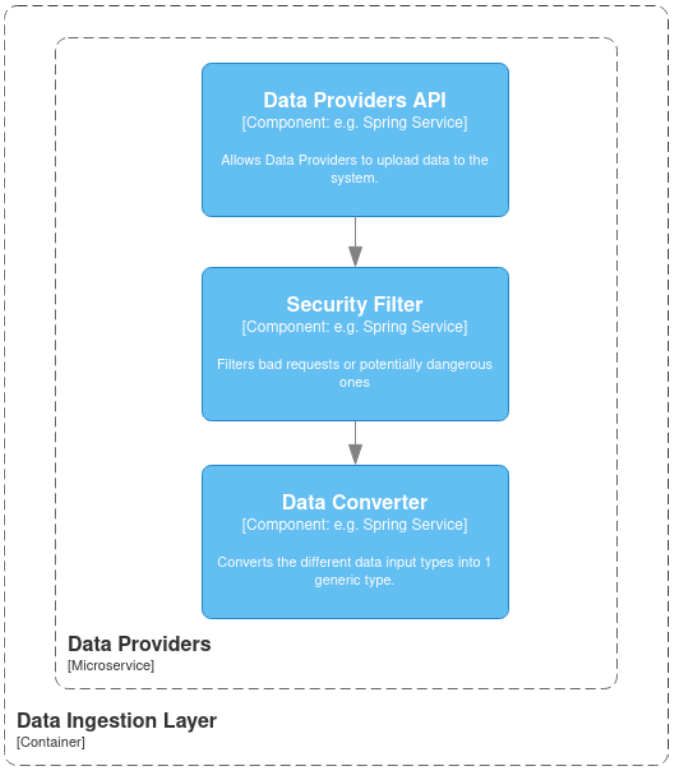

Data Providers API
[Component: e.g. Spring Service]
Allows Data Providers to upload data to the system.
Security Filter
[Component: e.g. Spring Service]
Filters bad requests or potentially dangerous ones
Data Converter
[Component: e.g. Spring Service]
Converts the different data input types into 1 generic type.
Data Providers
[Microservice]
Data Ingestion Layer
[Container]

Figure 9. Data Ingestion Layer for RailPulse (Component Diagram)

8.2 Data Processing Layer

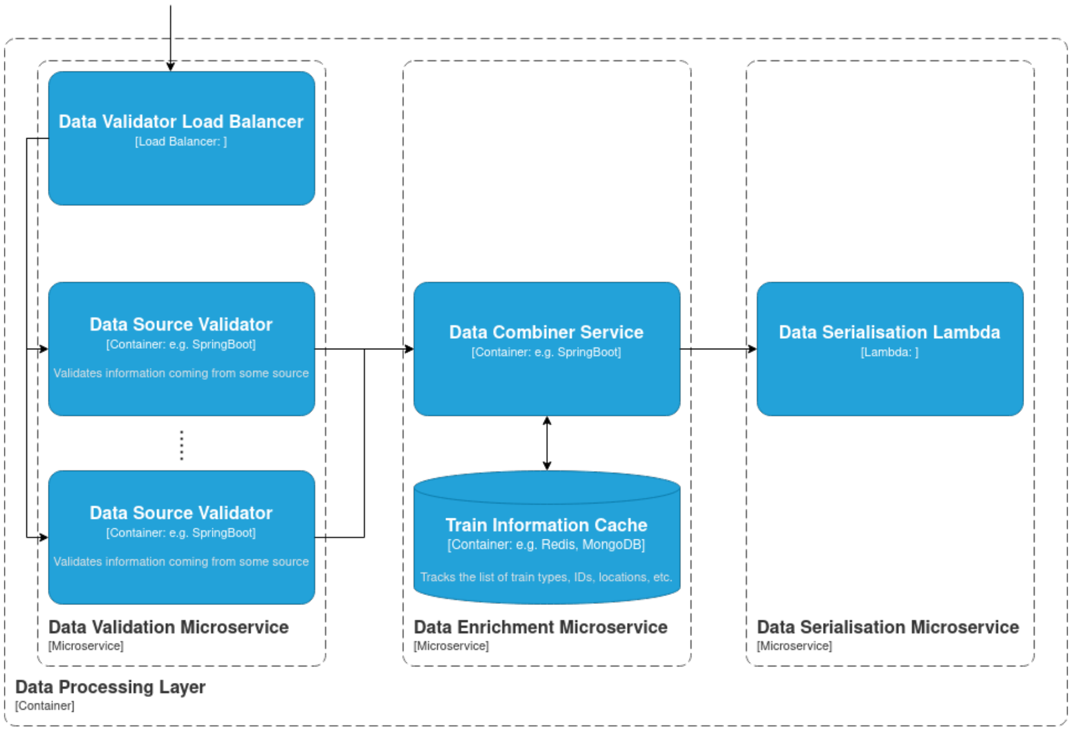

Data Validator Load Balancer
[Load Balancer:]
Data Source Validator
[Container: e.g. SpringBoot]
Validates information coming from some source
Data Source Validator
[Container: e.g. SpringBoot]
Validates information coming from some source
Data Validation Microservice
[Microservice]
Data Combiner Service
[Container: e.g. SpringBoot]
Train Information Cache
[Container: e.g. Redis, MongoDB]
Tracks the list of train types, IDs, locations, etc.
Data Enrichment Microservice
[Microservice]
Data Serialisation Lambda
[Lambda:]
Data Serialisation Microservice
[Microservice]
Data Processing Layer
[Container]

Figure 10. Data Processing Layer for RailPulse (Component Diagram)

The data processing layer seen in Figure 10 is responsible for validating, enriching, and serialising the input data. For each of these concerns, a microservice is constructed. For the validation, there is a validator container for each input source, which is chosen by the Data Validator Load Balancer. After the data is validated, it is combined in the Data Combiner Service, which needs to keep some information for later (such as train identification data), which is why it is accompanied by a Train Information Cache. Finally, if a 'full' data fragment is constructed from combining the data sources, it is serialised by the Data Serialisation Lambda, after which it moves on to the distribution layer. From a stakeholder perspective, the view describing the data processing layer could be used by multiple stakeholders such as: regulatory bodies, system architects or security teams, mostly to validate the system's compliance to legal and security regulations or as a reference for system development and maintenance.

8.3 Data Distribution Layer

8.3.1 Container View

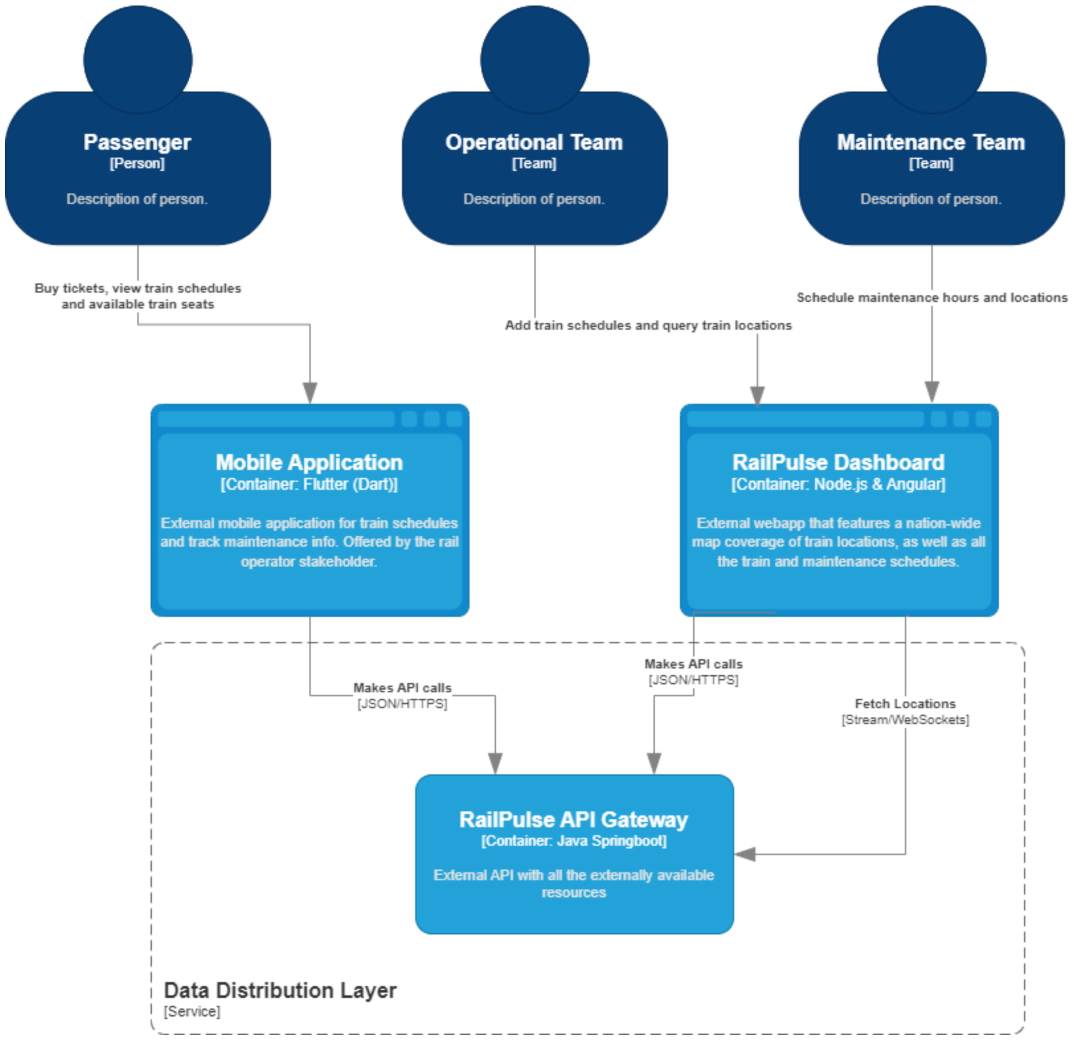

Passenger
[Person]
Description of person.
Operational Team
[Team]
Description of person.
Maintenance Team
[Team]
Description of person.
Buy tickets, view train schedules and available train seats
Add train schedules and query train locations
Schedule maintenance hours and locations
Mobile Application
[Container: Flutter (Dart)]
External mobile application for train schedules and track maintenance info. Offered by the rail operator stakeholder.
RailPulse Dashboard
[Container: Node.js & Angular]
External webapp that features a nation-wide map coverage of train locations, as well as all the train and maintenance schedules.
Makes API calls
[JSON/HTTPS]
Makes API calls
[JSON/HTTPS]
Fetch Locations
[Stream/WebSockets]
RailPulse API Gateway
[Container: Java Springboot]
External API with all the externally available resources
Data Distribution Layer
[Service]

Figure 11. Data Distribution Layer for RailPulse (Container Diagram)

The data distribution layer represents the last step in the data processing pipeline. Presented in Figure 11, the data distribution layer is externally visible in the form of an API gateway, which serves data to multiple stakeholders, such as passengers, who use external mobile applications to view train schedules, or operational and maintenance teams which use dashboards that show a live map of train locations, maintenance sites and more, with data extracted from the gateway. The API gateway that acts as the front door for the data distribution layer is serverless, which allows RailPulse to extend to multiple gateways, depending on each client's needs.

8.3.2 Component View

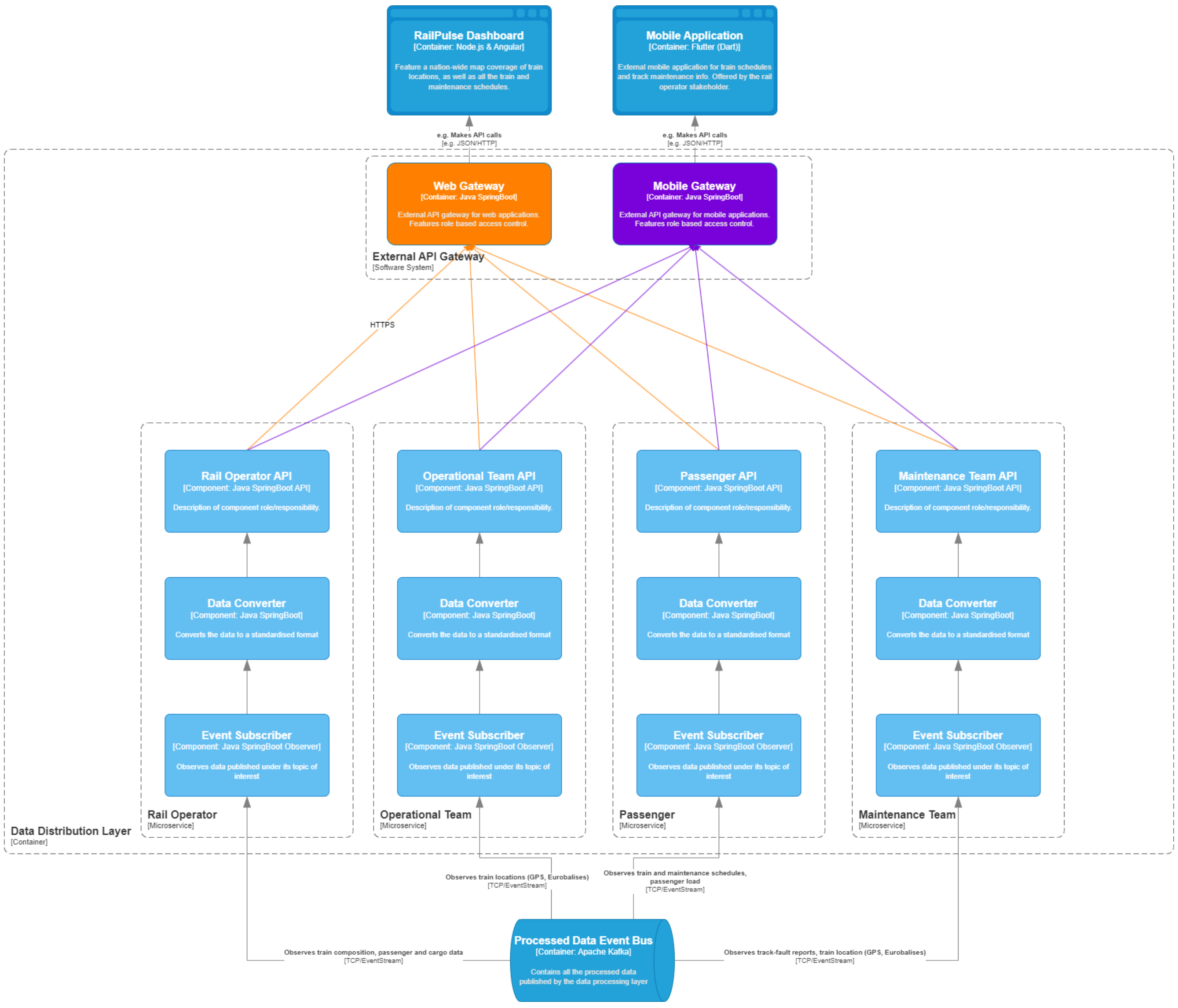

RailPulse Dashboard
[Container: Node.js & Angular]
Feature a nation-wide map coverage of train locations, as well as all the train and maintenance schedules.
Mobile Application
[Container: Flutter (Dart)]
External mobile application for train schedules and track maintenance info. Offered by the rail operator stakeholder.
e.g. Makes API calls
[e.g. JSON/HTTP]
e.g. Makes API calls
[e.g. JSON/HTTP]
Web Gateway
[Container: Java SpringBoot]
External API gateway for web applications. Features role based access control.
Mobile Gateway
[Container: Java SpringBoot]
External API gateway for mobile applications. Features role based access control.
External API Gateway
[Software System]
HTTPS
Rail Operator API
[Component: Java SpringBoot API]
Description of component role/responsibility.
Operational Team API
[Component: Java SpringBoot API]
Description of component role/responsibility.
Passenger API
[Component: Java SpringBoot API]
Description of component role/responsibility.
Maintenance Team API
[Component: Java SpringBoot API]
Description of component role/responsibility.
Data Converter
[Component: Java SpringBoot]
Converts the data to a standardised format
Event Subscriber
[Component: Java SpringBoot Observer]
Observes data published under its topic of interest
Rail Operator
[Microservice]
Operational Team
[Microservice]
Passenger
[Microservice]
Maintenance Team
[Microservice]
Data Distribution Layer
[Container]
Observes train locations (GPS, Eurobalises)
[TCP/EventStream]
Observes train and maintenance schedules, passenger load
[TCP/EventStream]
Observes train composition, passenger and cargo data
[TCP/EventStream]
Processed Data Event Bus
[Container: Apache Kafka]
Contains all the processed data published by the data processing layer
Observes track-fault reports, train location (GPS, Eurobalises)
[TCP/EventStream]

Figure 12. Data Distribution Layer for RailPulse (Component Diagram)

In the component view, the data distribution layer is described as a container which holds multiple subsystems, each encapsulated in one microservice. Each external system that interacts with the RailPulse API has specific data exchange formats and parameters. To mitigate this issue, the data distribution layer abstracts the external API gateway, offering a common interface for the clients. The abstract gateways are implemented under the hood using Lambdas, one for each type of data, or external service which needs custom handling of data. Three components can be found inside each microservice. The event subscriber is simply a serverless subscriber to a shared event bus which holds all the processed data published by the data processing layer. Whenever an event of interest is published, the subscriber consumes it, checks the validity of data, and then passes it to the data converter component. The data converter is simply a Lambda function which converts the data objects into the standardised format agreed upon with each external actor. The last component inside each microservice is an API, which implements the abstract external interface. By performing the task of consuming events, converting them to a standard exchange format and abstracting away API calls using serverless functions and gateways, we ensure that the data distribution layer of RailPulse can be quickly deployed, easily scaled and maintainable.

8.4 Auxiliary Layer

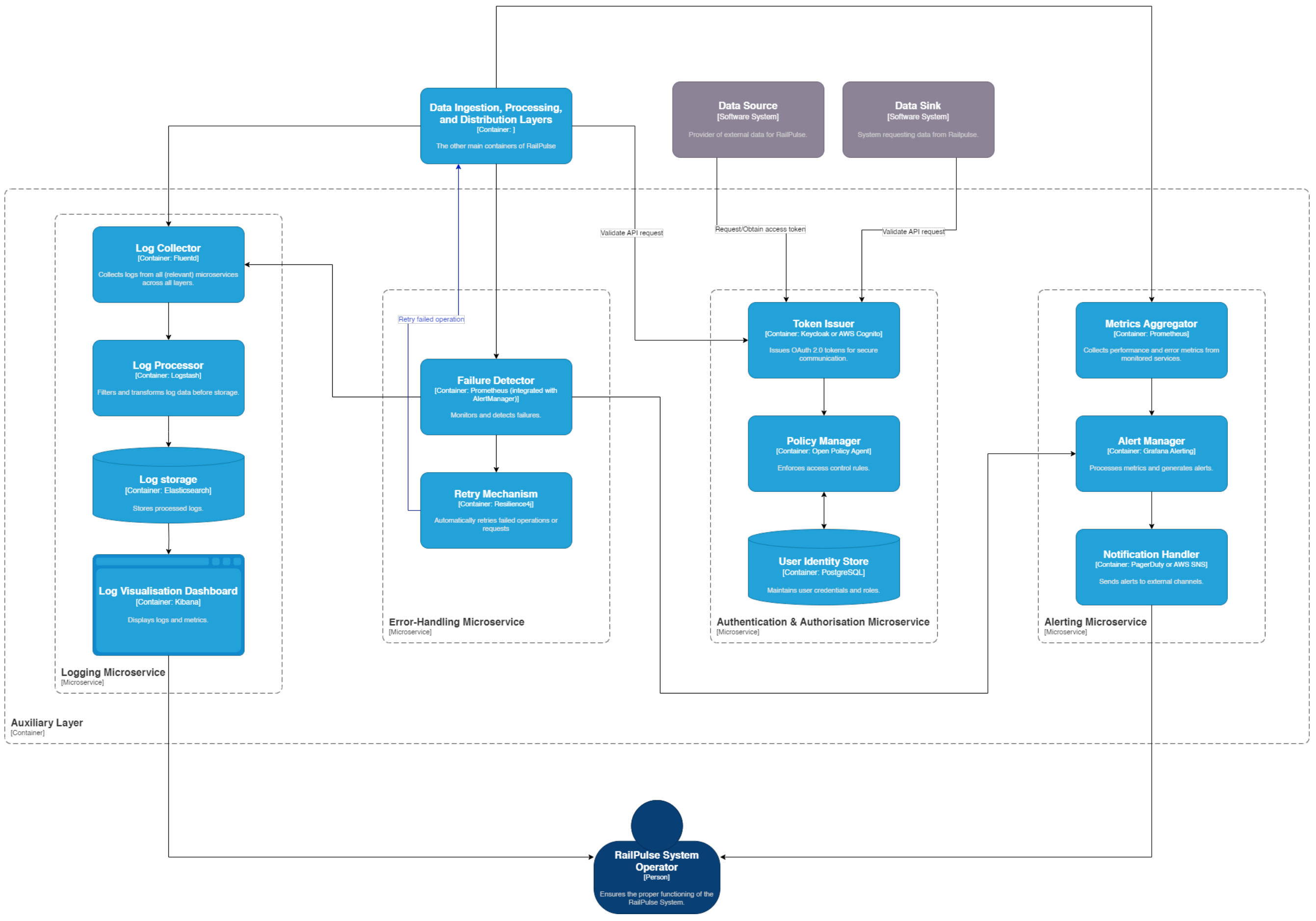

Data Ingestion, Processing, and Distribution Layers
[Container:]
The other main containers of RailPulse
Data Source
[Software System]
Provider of external data for RailPulse.
Data Sink
[Software System]
System requesting data from Railpulse.
Validate API request
Request/Obtain access token
Validate API request
Retry failed operation
Log Collector
[Container: Fluentd]
Collects logs from all (relevant) microservices across all layers.
Log Processor
[Container: Logstash]
Filters and transforms log data before storage.
Log storage
[Container: Elasticsearch]
Stores processed logs.
Log Visualisation Dashboard
[Container: Kibana]
Displays logs and metrics.
Logging Microservice
[Microservice]
Failure Detector
[Container: Prometheus (integrated with AlertManager)]
Monitors and detects failures.
Retry Mechanism
[Container: Resilience4j]
Automatically retries failed operations or requests
Error-Handling Microservice
[Microservice]
Token Issuer
[Container: Keycloak or AWS Cognito]
Issues OAuth 2.0 tokens for secure communication.
Policy Manager
[Container: Open Policy Agent]
Enforces access control rules.
User Identity Store
[Container: PostgreSQL]
Maintains user credentials and roles.
Authentication & Authorisation Microservice
[Microservice]
Metrics Aggregator
[Container: Prometheus]
Collects performance and error metrics from monitored services.
Alert Manager
[Container: Grafana Alerting]
Processes metrics and generates alerts.
Notification Handler
[Container: PagerDuty or AWS SNS]
Sends alerts to external channels.
Alerting Microservice
[Microservice]
Auxiliary Layer
[Container]
RailPulse System Operator
[Person]
Ensures the proper functioning of the RailPulse System.

Figure 13. Auxiliary Layer for RailPulse (Component Diagram)

The Auxiliary Layer in the RailPulse architecture is responsible for addressing cross-cutting concerns that support the operation, monitoring, and security of the system. It is designed to concurrently communicate with most other individual microservices of the system, ensuring that auxiliary tasks, such as logging, authentication, error handling, and alerting, are managed separately from the primary data pipeline. By functioning akin to a high-level ”daemon thread” it ensures that non-essential operations run in parallel with the main features of the system - enhancing scalability and maintainability - while also improving fault tolerance by having a centralised mechanism for error-handling.

The layer comprises several microservices, each focusing on a distinct function. The Logging Microservice collects and centralizes logs from all other layers, enabling system administrators to monitor performance, trace errors, and ensure compliance with regulatory requirements. It interacts with all system layers to aggregate logs into a centralised repository, facilitating analysis and troubleshooting. Moreover, it provides a dashboard interface that system operators can use to visualise key metrics such as system health, performance trends, and error rates - enabling them to identify anomalies or gain actionable insights to optimise system performance.

The Authentication and Authorisation Microservice actively communicates with the data ingestion and distribution layers, acting as a gatekeeper for all incoming and outgoing requests. It maintains system security and integrity by ensuring robust access control - integrating with an OAuth 2.0 server to authenticate users and services during their interactions with the system. By using a dedicated component, this microservice manages token issuance and lifecycle, such that tokens are securely generated, refreshed, and revoked when access privileges change. Assisting in this task is the Policy Manager component, which defines and enforces role-based access control (RBAC). This is done by obtaining user roles and their permissions from a User Identity Store and attributing them to the users with the appropriate credentials.

The Error Handling Microservice is designed to detect and mitigate faults within the system. By forwarding potential failure logs to the appropriate microservice and initiating fail-safe mechanisms, it ensures that system operations continue uninterrupted.

In critical cases where automated responses are insufficient, warnings are passed down to the Alerting Microservice which notifies system administrators in real-time via predefined communication channels. This ensures prompt human intervention for issues requiring immediate attention, such as system downtime or critical security breaches. Additionally, the alerting microservice can also generate warnings based on worrying trends in collected system metrics - helping to mitigate potential issues such as bottlenecks or high latency before they even occur.

From a stakeholder perspective, the auxiliary layer represents an object of great interest because it centralises important aspects of the overall architecture such as: authentication and authorization, error handlers, and logs. The authentication and authorization layer can therefore easily be analysed and audited by security teams or regulatory auditors. The error handling mechanisms combined with the logging functionality enables developers, security and DevOps teams to quickly trace, identify and solve problems in the least amount of time.

9 RELATED PROCESSES

One last major aspect influencing the success or failure of the RailPulse system is its Realisation. Being one of the three main system scopes, this project aspect shapes the structure of the development process by defining how the architecture transitions from concept to implementation. Doing this, it ensures that the architecture delivers the right solution to the right problem (i.e., meeting the technical requirements), while also fitting within the allocated development time and budget.

This section aims to explore the development processes related to the realisation of the RailPulse system, presenting a set of recommended methodologies and arguing how they, accompanied by relevant tactics and patterns, contribute to achieving the project's goals effectively and efficiently. In particularly, it explains why the overarching process is best suited to be a Waterfall process, while each section of this process follows a more Agile approach, as defined in the Agile manifesto [5] - not to be confused with particular implementations such as SCRUM or Extreme Programming. Finally, we argue why DevOps is the primary choice for the Deployment and Maintenance phases.

9.1 System Development and Operation - Waterfall

Considering the large size and, more importantly, the safety-critical nature of the RailPulse system, our team believes the Waterfall model is best suited to guide the overall development process. Its clear separation of project stages ensures that the scope remains well-defined and manageable, which is critical for a system of this complexity. For example, eliciting and validating requirements comprehensively at the start—rather than iteratively, as in Agile approaches, provides an approved and stable source of truth to guide development. This is particularly beneficial for large systems, where continuously redefining requirements during sprints would be inefficient and could lead to scope creep. Additionally, the dependencies and complex relationships between system components, such as aggregating data from multiple sources, enriching it, and distributing it to various sinks, make Agile ”test-as-you-go” approaches impractical. Finally, this complexity, along with the risk-sensitive nature of the system and associated risk implications, highlights the importance of system testing, and its priority over other, finer-grained testing stages (such as unit or integration testing). This aspect requires the full implementation of the system to be completed before verification, thus ensuring a comprehensive validation of safety-critical features, which are often dependent on each other or require fail-safe mechanisms triggered by the system as a whole.

Nevertheless, some might (rightfully) argue that the main drawback of the Waterfall model is its rigidity, particularly its linear progression, which can make adapting to unexpected changes challenging. However, for the RailPulse system, this rigidity is more of an advantage than a hindrance, as the clearly defined stages help maintain focus and accountability in a complex, safety-critical environment. Additionally, this drawback is mitigated by our recommendation to adopt a flexible approach within individual stages, such as incorporating Agile practices for tasks like requirements elicitation, design, and implementation. Importantly, we do not advocate for the use of any specific Agile methodology, such as SCRUM or Extreme Programming. Instead, we suggest an approach grounded in the Agile philosophy—one that emphasizes iterative efforts (whether for requirements elicitation, design, or implementation) coupled with continuous validation and re-iteration. This allows for incremental refinements and quality control within each phase, promoting adaptability and responsiveness to change. At the same time, the overall structure of the Waterfall model ensures that completed stages serve as stable foundations, minimizing the need for costly changes downstream. An overview of the recommended development approach (both high- and low-level) is depicted in Figure 14.

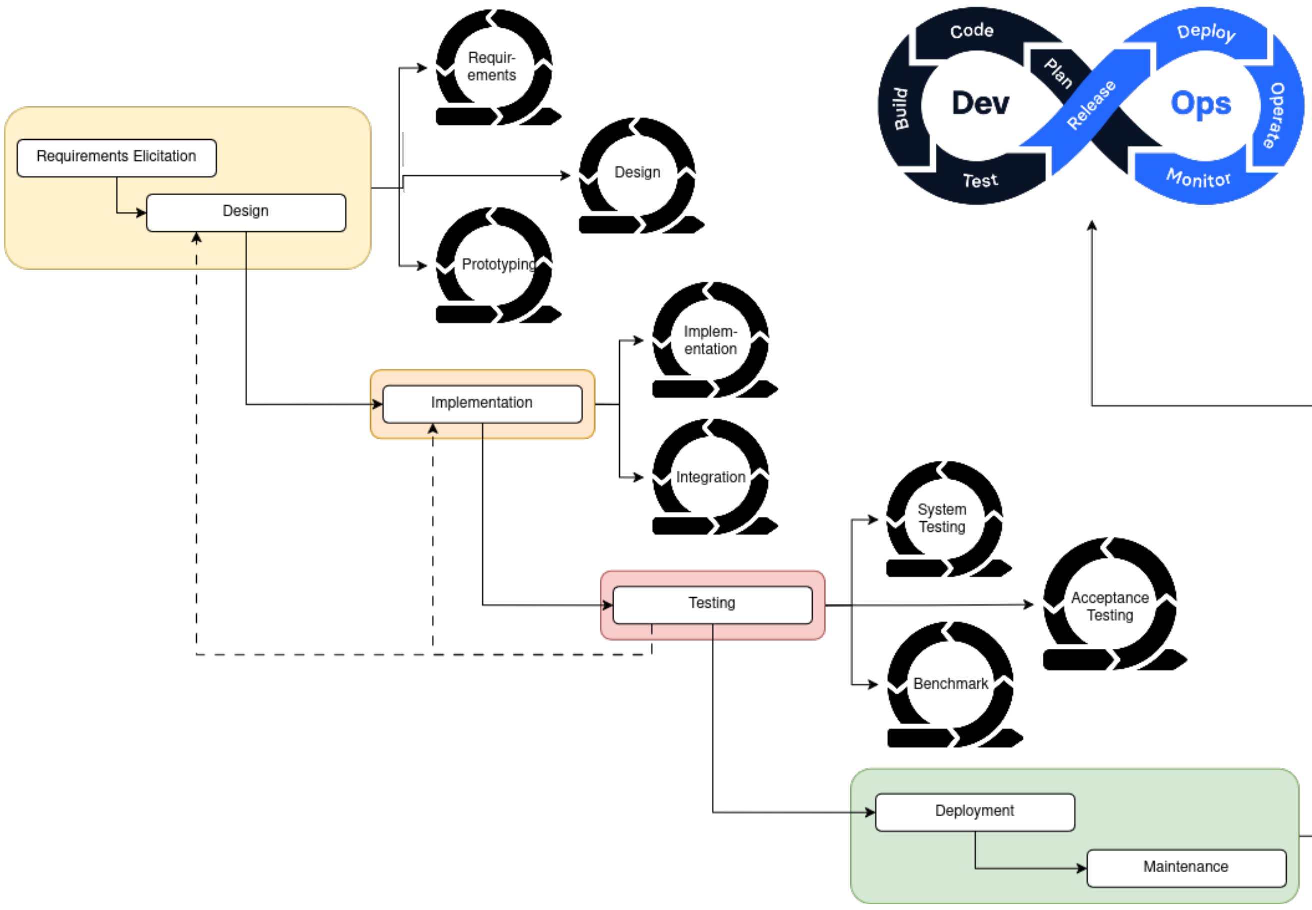


Figure 14. Recommended development structure for RailPulse. Waterfall model for high-level organisation; Agile for the first 4 stages of the model, and DevOps for the deployment & maintenance phases. Dotted lines represent optional choices (as one might discover new requirements or a missing implementation during testing).

9.2 Requirements Elicitation & Design - Agile

As previously stated, a major implication of the safety-critical nature and complexity of the RailPulse system is the necessity of establishing a comprehensive, (rationally) exhaustive, and validated list of requirements before any implementation can begin. This choice mitigates the risk of misunderstanding the system's goals and prevents costly re-implementation efforts, which would be challenging given the system's scale and intricate web of dependencies. Nevertheless, such proper requirements specifications cannot feasibly be obtained "in one go" - multiple iterative rounds of elicitation and validation are needed to ensure that all stakeholder needs are captured, acknowledged, and confirmed.

An Agile approach is particularly well-suited for this phase, as its iterative nature facilitates continuous stakeholder involvement and refinement. Recommended elicitation techniques include **document studies** and **interviews** to identify and formalise potential system objectives, complemented by **focus groups** to resolve conflicting stakeholder priorities. Additionally, **risk analysis** should be incorporated to ensure all safety-related concerns are identified and addressed in the implementation, while **cost-benefit analysis** aids in prioritising requirements effectively, ensuring the development plan aligns with the project's time and budget constraints (a critical verification considering the system's size and complexity).

For validation, we recommend **goal-domain analysis** to ensure that each goal is properly addressed and prioritised, as well as identify any potentially superfluous features (i.e., which are not associated with any system goal). This verification should be complemented by a **domain-requirements analysis**, which verifies whether all relevant quality attributes are properly formalised into requirements. A description of all these elicitation and validation techniques is provided in [18].

In adherence to the Waterfall model, the design stage should be distinct from the requirements elicitation phase. However, the close interplay between these two stages—more pronounced than between any other phases in the development life-cycle—necessitates a degree of flexibility. While the design phase should ideally begin only after requirements are thoroughly elicited and validated, we acknowledge the value of iterative feedback loops in this context. For instance, preliminary design steps, such as **system models** or **prototypes**, can assist in eliciting additional requirements or clarifying ambiguous ones. Similarly, during the design process, it may become evident that certain requirements need re-evaluation or that previously overlooked requirements emerge. For these reasons, we recommend an iterative exchange between requirements elicitation and system design, promoting an architecture which is accurate, robust, and accommodates the dynamic nature of stakeholder needs - especially at such an early stage in the development process.

9.3 System Implementation - Agile

For the implementation stage of the RailPulse system, we recognize that no single development methodology is universally superior. Nevertheless, we would still recommend an Agile approach due to its high level of adaptability and alignment with the project's requirements and architecture. Additionally, a more informal benefit of employing such a methodology is that we also recommend it for the "surrounding" development stages such as Requirements Elicitation, Design, and Testing. Hence, we realise that having to switch "back and forth" between various techniques might have a negative impact on the development's team throughput.

For a start, the main advantage of Agile in the implementation of the RailPulse system is its ability to handle changes flexibly, especially in the wider context of using the Waterfall model to guide overall system development. While high-level requirements are unlikely to change after stakeholder validation - as this process also involves the actors committing to the agreed-upon solution, technical or design-level requirements may evolve, particularly if development stretches out over a long period of time (as is to be expected with large-scale, complex projects). For example, interfacing systems may update their API gateways or data formats during the development timeline, in response to their own requirements and business environment. This unexpected development does not require a change in RailPulse's overarching goals or system design, albeit it does warrant a change in the implementation of a particular feature. In such scenarios, Agile's iterative nature is a perfect fit, allowing the development team to promptly address such changes within a subsequent sprint, ensuring minimal disruption to the underlying system structure.

From a more practical standpoint, Agile methods synergise notably well with the microservice-based architecture we recommend for RailPulse [12]. Individual sprints can focus on the complete development of specific microservices, including their integration into the existing system and verification of inter-component interactions. This modular approach ensures that each service is fully operational and validated before moving on to the next, promoting a robust and scalable system. Additionally, Agile's emphasis on incremental verification is particularly beneficial given the project's size and complexity. By testing newly implemented microservices or features during the same sprint (e.g., through unit and integration testing), developers can build a solid foundation for subsequent system tests, which we motivated as critical for the success of such a safety-critical product. This early validation ensures that the implementation process remains on track, reducing the likelihood of major issues arising during later development stages (testing, deployment, maintenance) and thus preventing additional costs from downstream bug fixes.

9.4 Testing - Agile

As was the case in the previous stages, we believe that having the Testing phase occur "naturally" within the bounds of the Waterfall methodology would be beneficial from a development standpoint. This way, the prioritised system tests can occur in a stable environment, in which all features have been implemented and validated to some degree, both internally as well as in relation with others. Nevertheless, RailPulse remains a large and complex product which cannot be properly verified in a single testing iteration, especially if we also take into account the various other systems it needs to interact with. To this end, we again propose an Agile approach for the organisation of the Testing stage, however with a caveat. Given the dependencies between system components and the prioritisation of system-wide functionality over isolated features, we recommend organising testing sprints not around specific subsystems to be tested but around comprehensive methods for evaluating the system as a whole. For instance, three possible testing sprints are given below:

9.4.1 System Testing

The first testing sprint should focus on **technical system tests**, which use formal techniques as well as observations in a controlled environment to check if the RailPulse system meets the goals outlined during the Requirements Elicitation phase. When it comes to formal verification, the testers can employ System Validation techniques, such as bounded analysis, static verification, and runtime assertion checking to identify potential logic errors or operational inconsistencies. Additionally, testing could involve observing system performance within a controlled demo environment, taking notes of any undesirable behaviour. If successful, the scope of such observation tests can be expanded and the system be deployed as a pilot on a limited section of rail infrastructure. This way, the testing team can verify system behaviour in a realistic setting, ensuring that all core functionalities operate as intended before scaling to full deployment. Additionally, such pilots serve as the basis of acceptance testing, which can be the focus of a subsequent sprint.

9.4.2 Acceptance Testing
As stated previously, a suitable successor to the technical system testing sprint would be a user acceptance one, involving all data sources and sinks in the production environment. The goal is to ensure that RailPulse interfaces efficiently with external systems, be they data sources (e.g., regional dispatch zones, GPS providers, etc.) or data sinks (e.g., rail operator systems, passenger information platforms, etc.). This stage tests whether data ingestion, processing, and distribution occur as designed, verifying that RailPulse integrates effectively with existing workflows while also meeting the imposed quality requirements. Particular attention should be given to how well data sinks can utilise the information provided by RailPulse, ensuring that outputs are both accurate and usable.

9.4.3 Benchmarking Quality Attributes
Another major testing sprint could focus on benchmarking the system against the prioritised quality attributes, using the response measures detailed in Section 6 and additional industry standards. This includes assessing system performance, availability, and safety under realistic operating conditions - like the ones present in a pilot deployment. For example, tests should verify system uptime targets, validate failover mechanisms, and measure latency and throughput during peak loads. These benchmarks provide quantifiable evidence of the system's adherence to stakeholder requirements and identify any areas needing optimization before deployment.

9.5 Deployment & Maintenance - DevOps
Continuing the trend of high-level adherence to the Waterfall model and a finer-grained, iterative approach internally, we recommend that the Deployment and Maintenance phases of RailPulse's development follow the DevOps methodology. In this way, these last development stages occur only after the implementation and testing phases are fully completed, preserving the stability and reliability expected from such a complex system. Internally, however, DevOps practices provide the flexibility and responsiveness necessary for the rapid rollout of updates and fixes, which is crucial given the operational demands and safety implications of the RailPulse system.

For a start, DevOps is particularly well-suited for deployment due to its streamlined processes, emphasis on automation, and support for both "scheduled" and "emergency" releases. Automated pipelines for continuous integration and delivery (CI/CD) reduce human error by simplifying tasks such as building, testing, and deploying code, ensuring consistency across environments. In addition, Infrastructure as Code (IaC) [19] can be employed to define and manage resources programmatically, helping RailPulse maintain consistent configurations between staging and production while accelerating environment setup or scaling. Following the DevOps methodology, these techniques should be employed both during initial system deployment as well as for major updates - which should adhere to a "scheduled release" model planned in collaboration with primary stakeholders. This way, all relevant parties can agree on a set of deliverables and their deadlines, helping to manage operational resources, as well as expectations. Nevertheless, DevOps also supports an "emergency release" approach which ensures that critical patches can be delivered swiftly to maintain system integrity — an essential consideration for a safety-critical system. For example, when urgent fixes are needed, development teams have the autonomy to quickly investigate and fix the problem, test the solution and ultimately deploy it to the production environment - without the "bureaucracy" required by more rigid methodologies. Ultimately, this allows for critical issues to be patched as soon as possible, helping to maintain the high level of safety and reliability prioritised by the system.

DevOps also brings notable advantages to the maintenance phase. The methodology's inherent adaptability allows for the quick implementation of changes, such as feature updates or bug fixes, while ensuring adequate testing, necessary for maintaining system integrity. Professional experience of the authors suggests that many DevOps environments operate with a parallel demo or staging application, enabling updates to be tested thoroughly before being pushed to production. This approach aligns with the strict quality requirements of RailPulse, allowing for fast-paced iterative improvements without compromising safety or reliability. Moreover, the continuous monitoring and logging practices central to DevOps provide real-time insights into system performance and health, enabling proactive identification and resolution of potential issues.

Beyond these technical benefits, such a flexible methodology is also strengthened by the collaborative practices associated with it. By fostering close cooperation among development, operations, and quality assurance teams, DevOps encourages cross-functional problem-solving and rapid feedback loops. In the context of RailPulse, such practices can be operationalised through regular stand-ups/post-mortems, shared monitoring dashboards, and integrated communication tools - all contributing to streamline incident response, facilitate timely stakeholder feedback, and ensure that improvements or fixes are quickly validated in staging before being released to production. Ultimately, such a collaborative and flexible approach to deployment and maintenance is especially suitable for a safety-critical, highly demanding operation environment such as the one described in this paper.

10 RISKS

In this section, we will discuss the key risks in the RailPluse project, their potential impacts and how we try and plan on reducing or mitigating them.

10.1 Invalid input data

RailPulse depends on several external data sources. Invalid or corrupted data can result in a compromise of the system's accuracy and/or reliability. This can happen because of many factors, including incorrectly configured or malfunctioning sensors, network errors or even malicious data injection. In order to try and mitigate this risk as much as possible, RailPulse will deploy data validation techniques and real-time error-checking mechanisms. For example, we will prioritise certain data points over others. Take for example the case of GPS data versus data from local dispatch zones: GPS data is usually more precise, so we would prefer this, however, we will verify this GPS data against the regional dispatch zone data to discard any data from malfunctioning sensors. In such a case, a warning message will be sent to the Maintenance Team so they can check on the situation.

10.2 Increased Latency due to Networking

Another risk is latency, since we plan on using multiple micro-services the communication between these services and the communication to the data providers add delay to the processing of data through our system. This delay, however, should be minimal and not an issue for our use case, while we need semi-real-time processing of the data, a delay of up to a few seconds does not cause any issues. However, to ensure timely processing of critical events we plan to implement priority-based queues and advanced message-routing strategies.

10.3 Reliance on third-party systems

Since we are abstracting away from infrastructure management by using a server-less approach we do introduce new risks. We are dependent on hardware that is not ours, since we are using third-party systems, while this has the advantage of being easy to deploy it can cause disruption when problems occur with the systems. We plan on reducing this risk by making the critical components of the system redundant, therefore in case something happens to our infrastructure provider, we can switch within 30 seconds to an alternative provider and route the traffic through there.

10.4 Security Concerns

Due to the critical nature of the RailPulse system, security vulnerabilities can have a large impact and cause substantial issues for large groups of people. It is therefore essential that, especially on the data ingestion side, we try and mitigate these risks by protecting against bad requests and having strict access controls. Additionally, the data flowing through the cloud-managed infrastructure requires robust encryption and access control measures to prevent breaches.

10.5 Integration Challenges

Integrating RailPulse with existing rail systems and technologies used may not always be as straightforward, therefore we will introduce a standardized API. Additionally, we involve the stakeholders from early on and do thorough testing during the development phase to improve integration.

10.6 Cost Overruns

Large-scale projects, like RailPulse, are prone to overrunning the budget due to unforeseen challenges and circumstances. In order to reduce these risks we deploy agile methodologies and check progress regularly. This ensures we prioritize the right features and address components of high importance first.

11 SPECIFICATIONS

This section outlines the external specifications referenced throughout the architecture of RailPulse, their sources, and how they were used to inform the system's design and implementation. The overview not only includes the specifications which shaped the structure of the architecture but also the ones which inspired certain design decisions.

ISO 42010 - General Architecture Description [14]

This standard provides the foundation for RailPulse's architecture. It shapes the documentation of the system's scope, stakeholders and their concerns, and the views/viewpoints necessary to support a coherent and consistent architectural narrative.

Volere Requirements Specification Template [25]

This template guided the identification, documentation, and classification of stakeholder roles and relationships using the stakeholder onion layering technique.

ISO 25010 - Quality Attributes [13]

The standard used to define and prioritise the quality attributes critical to the RailPulse system. Additionally, it guided the identification of appropriate response measures for each quality attribute to ensure the architecture met stakeholder requirements.

Quality Attributes Technical Report - CMU/SEI-95-TR-021 [2]

Technical report which assisted in the definition and scoping of quality attributes, as well as in the derivation of finer-grained, quality sub-attributes (necessary for the utility tree and scenarios).

C4 Model - Architectural Structure of Views [6]

Model used to structure the architectural views in a hierarchical approach, helping to transition from a high-level system context to detailed component diagrams as well as illustrate relationships and interactions among system components.

OpenAPI Specification [9]

Specification that inspired the design of RailPulse's interaction with external systems, assisting in the formalisation of the data ingestion and distribution layers. By providing a framework for defining standardised APIs, it helps define these components in a way which promotes integrability with external actors/systems.

Istio Service Mesh Specification [11]

Istio was used as a reference for designing the communication between microservices within each layer of the architecture. Its service mesh specification influenced decisions on routing, load balancing, and secure communication.

Kubernetes - Container Orchestration Standard [17]

Standard which inspired the modular architecture of RailPulse - based on container orchestration of microservices. This specification did not structure the architecture itself, instead it served as a proof of concept which justified certain design decisions regarding the usage of microservices.

OAuth 2 - Security Protocol [22]

This specification inspired security-related architectural decisions for the RailPulse system. It provided the framework necessary to structure secure communication among microservices and between microservices and external actors.

AWS Lambda and API Gateways [29]

The AWS Lambda and API Gateway specifications provided a proof of concept for the feasibility of integrating microservice and serverless-based solutions within the RailPulse architecture. These specifications helped validate the scalability and cost-efficiency of the serverless design approach for handling fluctuating workloads, affirming its suitability to meet the system's dynamic resource allocation and communication requirements.

12 CONCLUSION

This document presents a comprehensive architecture for the RailPulse system, a real-time train monitoring solution designed to modernise European rail networks. RailPulse addresses the growing need for reliable, accurate, and timely train positioning data across national rail networks, enabling efficient scheduling, enhanced operational efficiency, and improved passenger satisfaction. By collecting train position data from regional dispatch zones and integrating supplemental data from external sources such as GPS and geolocation systems, RailPulse provides a unified, nationwide service that supports safe, punctual, and coordinated train operations.

Based on the system's mission-critical requirements of availability, performance, and safety, a microservice-based approach is proposed, based on its advantages in relevant concerns for key stakeholders. The designated layers for Data Ingestion, Processing, and Distribution - as well as the Auxiliary layer for logging and error handling - promote a modular, highly scalable, yet clear solution, essential for a safety-critical system such as RailPulse. The use of serverless functions and an event-driven architecture further enhance the system's responsiveness - allowing for handling fluctuating workloads as well as unexpected circumstances.

The architecture is complemented by the suggestion of a hybrid development approach, combining the structured, linear progression of the Waterfall model with the iterative, flexible practices of Agile methodologies. This way, the prioritised safety requirement is met by ensuring a careful and thorough development, with each waterfall step serving as the foundation for the rest, while also maintaining a high degree of flexibility in the face of changes - inevitable for such a complex business environment. On the same note, DevOps practices are recommended for deployment and maintenance, promoting a fast and proactive approach to dealing with bugs which might endanger the system's integrity or availability.

In conclusion, the RailPulse system represents a significant step forward in the modernisation of European rail networks by providing reliable, accurate, and real-time train positioning data. This architecture document serves as a foundational reference for the system's development, ensuring that RailPulse meets its goals and delivers lasting value to all stakeholders.

REFERENCES

[1] AWS (2024). https://aws.amazon.com/compare/the-difference-between-monolithic-and-microservice

[2] Barbacci, M., Klein, M., Longstaff, T., and Weinstock, C. (1995). Quality attributes. Technical Report CMU/SEI-95-TR-021, Carnegie Mellon University. Accessed: 2024-Dec-14.

[3] Basavegowda Ramu, V. (2023). Performance impact of microservices architecture. *The Review of Contemporary Scientific and Academic Studies*, 3.

[4] Bass, L., Clements, P., and Kazman, R. (2021). *Software Architecture in Practice, 4th Edition*. Addison-Wesley Professional. Accessed: 2024-Dec-5.

[5] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. (2001). http://agilemanifesto.org/.

[6] Brown, S. (2024). C4 model for visualising software architecture. https://c4model.com.

[7] EU (2024). ERTMS in brief.

[8] Hunt, A. and Thomas, D. (2000). The pragmatic programmer: From journeyman to master. In *The Pragmatic Programmer: From Journeyman to Master*, chapter 7, pages 242–263. Addison-Wesley Longman Publishing Co., Inc., USA.

[9] Initiative, O. (2021). Openapi specification. https://swagger.io/specification/.

[10] Intertech (2024). Train positioning system (tps) using rfid system.

[11] Istio (2024). Istio service mesh documentation. https://istio.io/.

[12] Jenkins, M. and Scalise, A. (2024). Microservices: An agile infrastructure for an agile world. *85th EAGE Annual Conference & Exhibition (including the Workshop Programme)*.

[13] JTC1/SC7, T. C. I. (2011a). Iso/iec 25010:2011. *ISO*.

[14] JTC1/SC7, T. C. I. (2011b). Iso/iec/ieee 42010:2022. *ISO*.

[15] Kazman, R., Klein, M., and Clements, P. (2000). Atam: Method for architecture evaluation. Technical Report CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA.

[16] Kruchten, P. (1995). Architectural blueprints — the "4+1" view model of software architecture. *IEEE Software 12*.

[17] Kubernetes (2024). Kubernetes documentation. https://kubernetes.io/.

[18] Lauesen, S. (2001). *Software Requirements: Styles and Techniques*. Pearson Education, 1 edition.

[19] Microsoft Learn (2025). What is infrastructure as code? https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code. Accessed: 2025-01-04.

[20] NIST (2025). Cybersecurity — nist. https://www.nist.gov/cybersecurity.

[21] NS (2024). Planned engineering work | NS. https://www.ns.nl/en/travel-information/maintenance-on-the-tracks/.

[22] Okta (2012). Oauth 2.0 authorization framework. https://oauth.net/2/.

[23] Olson, D. (2013). Stakeholder communications matrix. http://www.bawiki.com/wiki/Stakeholder-Communications-Matrix.html.

[24] OWASP (2025). Owasp top 10. https://owasp.org/www-project-top-ten/.

[25] Robertson, S. and Robertson, J. (2000). Volere requirements specification template.

[26] Services, A. W. (2024a). Aws ec2 container. https://aws.amazon.com/ec2/.

[27] Services, A. W. (2024b). Aws event bridge. https://aws.amazon.com/eventbridge/.

[28] Services, A. W. (2024c). Aws lambda. https://aws.amazon.com/lambda/.

[29] Services, A. W. (2024d). Aws lambda documentation. https://aws.amazon.com/documentation/.

[30] Shilkov, M. (2021). https://mikhail.io/serverless/coldstarts/aws/.

[31] Thales (2022). Rolling lab will gather data to enhance railway safety | Thales Group. https://www.thalesgroup.com/en/united-kingdom/news/rolling-lab-will-gather-data-enhance-railway-safety.

APPENDICES

A WORK DISTRIBUTION

Table 1. Teamwork distribution

Part of Document	Teammate
Scope and Introduction, System Context Diagram	Vlad Gosa
Related systems, trends and developments	Vlad Gosa
Stakeholder definition and stakeholder prioritization matrix	Everyone
Stakeholder communication	Bjorn Vuurens
Concern & Drivers	Douwe Osinga
Quality Attributes	Victor Alecu, Douwe Osinga, Bjorn Vuurens
Data Flow and Deployment View	Vlad Gosa
Logical View	Victor Alecu
Architectural Patterns	Victor Alecu, Douwe Osinga, Bjorn Vuurens
Component Specific Views	Everyone
Related Processes	Victor Alecu
Risks	Bjorn Vuurens
Specifications	Victor Alecu

B CONCERNS AND STAKEHOLDERS TABLES

Table 2. Prioritised Stakeholders

Priority	Stakeholder	Motivation
1	RailPulse Inc.	Needs the product to succeed in order to guarantee financial gains and increased market share.
2	European Union (EU)	Finances the project to enhance trade within Europe and boost the economy.
3	Rail Operators	Directly manage and operate train services, of which the operational efficiency, scheduling, punctuality, and overall service reliability will be enhanced by the RailPulse system.
4	Operational Teams (Dispatchers, Controllers)	Are responsible for managing train movements and ensuring smooth operation of the rail network. They rely on accurate real-time data to make decisions regarding scheduling, routing and handling unexpected events.
5	Freight Customers	Rely on the rail network for the timely and efficient delivery of their goods.
6	Passengers	Will indirectly notice the impact on efficiency, scheduling, punctuality, and overall service reliability that RailPulse offers to its business customers.
7	Railway Maintenance Team	Is responsible for maintaining the infrastructure and repairing the rolling stock.
8	Railway Infrastructure Owners	The organisation responsible for maintaining the infrastructure.
9	Data Providers (GPS Providers)	Supply the core data that the RailPulse system relies upon, as well as other supplemental data to enrich its services.
10	Competitors	Will lose financial and market share if RailPulse gains popularity, but may also use public information available from RailPulse to better their own services.
11	Software DevOps Team	Develop and maintain system, are being paid to ensure system keeps functioning.
12	Regulatory Bodies	Have a direct interest in ensuring that the system operates safely, complies with existing legislation, and maintains fair competition, to help the economy and to mitigate security and public safety issues.
13	Security Team	Are paid to ensure the security of the system and its adherence to data protection laws such as GDPR.
14	System Architects	Are a stakeholder, since they design the system.

Table 3. Prioritised Concerns

Priority	Concern	Classification*	Stakeholder(s)
1	Data Integrity	Q	All stakeholders
2	Optimise train scheduling	R	Rail Operators, Passengers, Freight Customers, EU
3	System Reliability	Q	Operational Teams, Passengers, Freight Customers, Railpulse Inc., Rail Operators, Software DevOps Team, EU
4	Fault Tolerance and Error Handling	R	Software DevOps Team
5	Minimal System Downtime	Q	Software DevOps Team
6	Timely Notification of Failures	Q	Software DevOps Team
7	Integration with Existing Systems	R,Q,D	All stakeholders / DevOps Team, Data Provider
8	Sensitive Data Protection	R,Q	All stakeholders
9	Regulatory Compliance	B,D,R	Rail Operators, Regulatory Bodies, EU, Railpulse Inc.**
10	Reputation and Customer Satisfaction	B	RailPulse Inc.**, EU
11	Product Differentiation	B	Competitors, RailPulse Inc.**
12	Low-latency, Real-time Decision making	Q	Operational Teams
13	Modularity	Q	Software DevOps Team, Architects
14	Scalability, Future Growth	B,Q	Software DevOps Team, Railpulse Inc.**, EU, Architects
15	Low Resource Utilisation ($\rightarrow$ Cost Savings)	B,D,Q	Data Providers, Software DevOps Team, Railpulse Inc.**, Architects
16	Cybersecurity Threat Mitigation	D,R,Q	Architects, Security Team, DevOps Team, RailPulse Inc.**, EU
17	Maintaining Market Share	B	RailPulse Inc.**, Competitors

*Business Goal (B), Quality Attribute (Q), Requirement (R), Domain Aspect (D)
**Railpulse Inc. = Company that owns Railpulse

C QUALITY ATTRIBUTES - UTILITY TREE

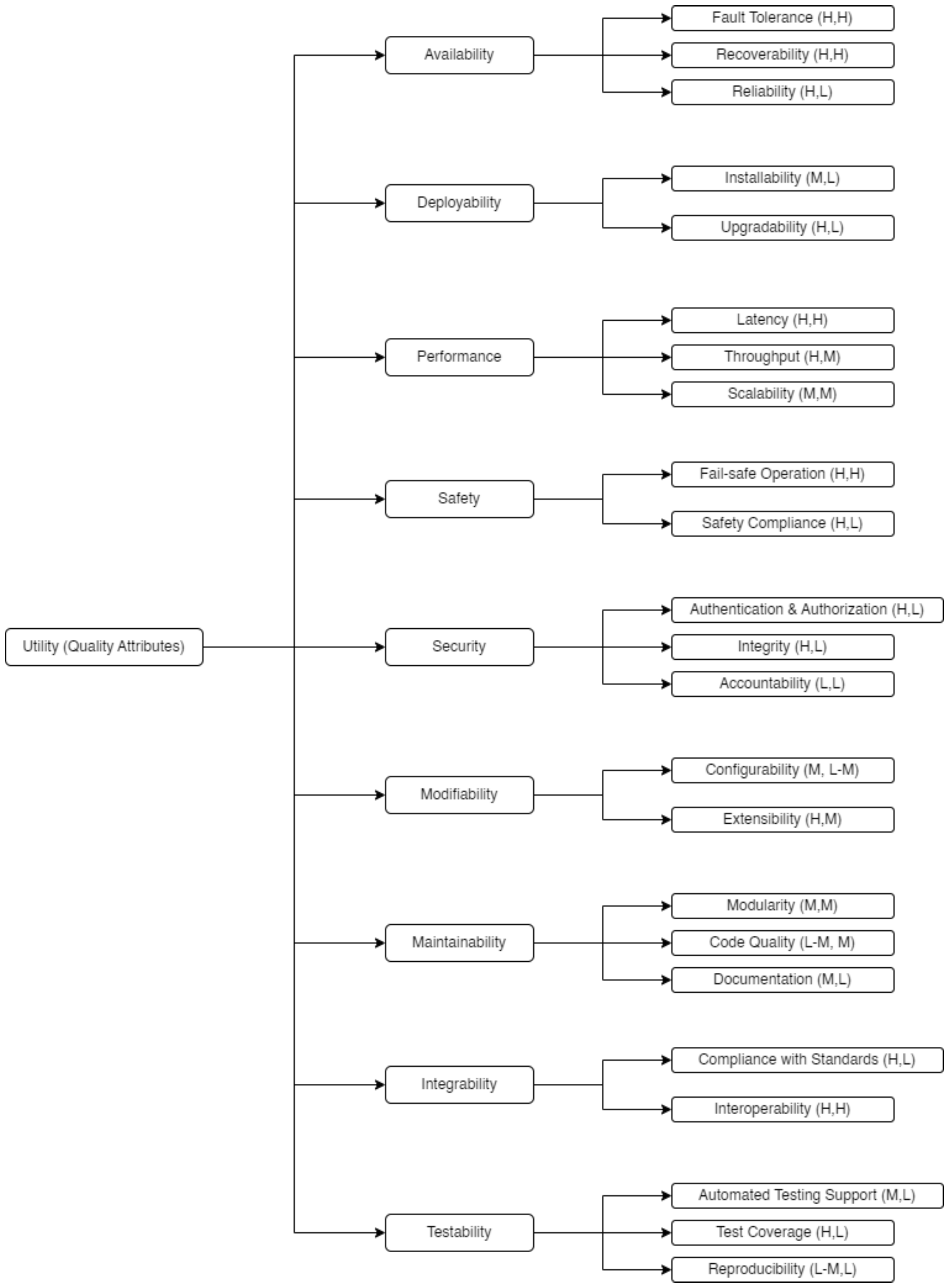


Figure 15. Graph describing different specification levels of quality attributes - inspired from Bass et al. (2021), Figure 14.1 [4]