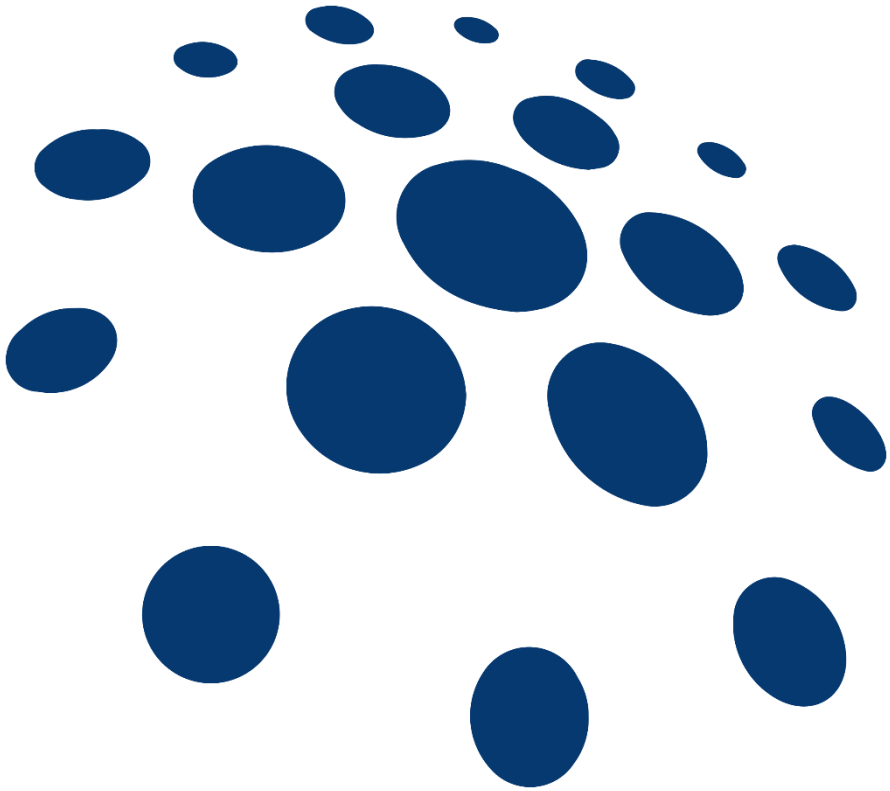
Horizon Europe

Project: 101138405 — Net4Cities

Deliverable 4.1

Systems Architecture



Net4Cities Consortium


Imprint

Suggested citation:
Ødegård, R., Chatterjee, A., Logna, R., Quedenau, J., Garcia, P. Sanz Pozo, R., Soares, J., Lopez-Aparicio, S. (2024).
D4.1 System Architecture. Horizon Europe Project Net4Cities.

Main findings and deliverables of the Net4Cities project will be available at www.net4cities.eu

This project has received funding from the European Union's Horizon Europe funding programme under the call HORIZON-CL5-2023-D5-01 – No. 101138405

Document Control Information

Settings	Value
Document Title:	D4.1 System Architecture.
Project Title:	Net4Cities
Document Author:	Rune Ødegård Ayan Chatterjee Robert Logna
Doc. Version:	1
Sensitivity:	Public
Date:	30/08/2024

Document Approver(s) and Reviewer(s):

NOTE: All Approvers are required. Records of each approver must be maintained. All Reviewers in the list are considered required unless explicitly listed as Optional.

Name	Role	Action	Date
Sanz Pozo, Roberto	Reviewer	Suggestions and comments that improved the document	22.07.2024
Jörn Quedenau	Reviewer	Suggestions and comments that improved the document	26.07.2024
Joana Soares	Reviewer	Suggestions and comments that improved the document	30.07.2024
Susana Lopez-Aparicio	Reviewer	Suggestions and comments that improved the document	07.08.2024
Erika von Schneidemesser	Review and Approval	Minor edits, review and approval	26.08.2024

Document history:

The Document Author is authorized to make the following types of changes to the document without requiring that the document be re-approved:

- Editorial, formatting, and spelling
- Clarification

To request a change to this document, contact the Document Author or Owner.
Changes to this document are summarized in the following table in reverse chronological order (latest version first).

Revision	Date	Created by	Short Description of Changes

Configuration Management: Document Location

The latest version of this controlled document is stored in the Sync&Share file-sharing platform at “Net4Cities/WP4/D4.1 systems architecture system architecture”

Table of Contents

List of Figures

List of Tables .....

Executive Summary

This document outlines the proposed architecture aiming to unify all parties involved and streamline the development process effectively. Its main objectives include offering developers' guidance and providing a framework for stakeholders to assess decisions and documenting key architectural choices for transparency and accountability. The document covers a scope, including a high-level overview of the intended system architecture and detailed descriptions of its components. It also provides a description of the framework employed.

The Net4Cities Data Hub is specifically designed to manage and analyze air pollution and noise-related data from sources such as urban air quality models, noise model systems, traffic data, and air quality and noise monitoring stations. This comprehensive platform integrates diverse data streams with advanced analytical models to deliver valuable insights into urban environmental dynamics. The system is structured into four layers: Data Ingestion, Data Storage, Data Processing and Querying, and Data Serving and Visualization. Each layer employs cutting-edge technologies to ensure data integrity, consistency, and accessibility. By bringing these elements together, the platform empowers decision-makers, enhances public awareness, and provides essential insights to researchers, policymakers, and citizens, facilitating informed actions for sustainable urban development.

Key technologies that are utilized include Kafka for real-time data streaming, Avro, for data serialization, Schema Server for managing data formats, Spark for processing large scale data, PostgreSQL for storing inventory data (metadata) and HBase for temporal and spatial observation data and model related data.

The operation of the system involves a series of steps starting with data ingestion. During this phase data is gathered from sources, validated and serialized. Subsequently this information is stored in storage solutions, processed using tools, like Apache Spark and GeoMesa before being accessed through a range of interfaces and visualization tools. This architectural approach ensures handling of data types on a large scale facilitating complex queries, real-time processing and interactive analysis of the data.

1. Introduction and Goals

Welcome to the Architecture Design Document for the Net4Cities Data Hub. This document acts as a blueprint of the systems architecture offering consortium members and stakeholders an overview and insight into the architectural strategies and decisions that will drive the development of the Net4Cities Data Hub. The main goal of this document is to ensure coherence among project contributors and facilitate communication throughout the development phase.

1.2 Purpose

The Architecture Design Document serves three purposes:

- **Guidance**: It provides structured descriptions of the system architecture to guide development teams outlining components, relationships and dynamic interactions. This includes defining software layers, subsystems, interfaces and dependencies.
- **Evaluation**: It establishes a framework for stakeholders to evaluate choices against requirements and constraints, ensuring that the architecture aligns with all specified business and technical objectives.
- **Documentation**: It documents decisions impacting the system providing argumentation, for choices made to promote clarity and accountability.

1.3 Scope

This document outlines the design of the system covering:

- Overview of the components and how they interact.
- Detailed description of each component's functions, capabilities and interactions.
- Specifications of the technology stack including platforms, programming languages and tools used.

The target audience includes:

- Developers working on Net4Cities Data Hub: To understand the system structure and their roles in implementing and integrating components.
- Quality Assurance Teams: To develop testing strategies based on the design and interconnections between components.
- Project Managers and Decision Makers: To monitor project progress and make informed decisions on resource allocation and risk management.
- External stakeholders: Such as clients and partners who require an understanding of the system's architecture for integration and evaluation purposes.

1.4 Structure of the Document

This document is divided into sections:

- Introduction: Providing an overview of the document's purpose, scope and target audience.
- Architectural Representation: Describing the model and design utilized.
- System Architecture: Offering an analysis of system components, their interconnections and the surrounding environment.
- Architectural Goals and Constraints: Clarifying the objectives and the limitations influencing the system design.
- Design Decisions: Summarizing choices made and explaining the reasoning behind them.

- Quality Scenarios: Outlining specifications for quality attributes such as scalability, reliability and maintainability.
- Appendices and References: Including information that supports decisions and references to external documents.

This document builds upon the open source Arc42 template[1], for documenting software and system architectures while adhering to the terminology and guidelines outlined in the standard ISO/IEC/IEEE 42010:2022.

2. Requirements Overview

The overall requirement of the Net4Cities Data Hub is to receive, store, process, provide and handle air pollution and noise related data, including emission inventories and metadata. The simplified concept diagram below (see Figure 1) illustrates the aggregate data requirement, data flow and end users.

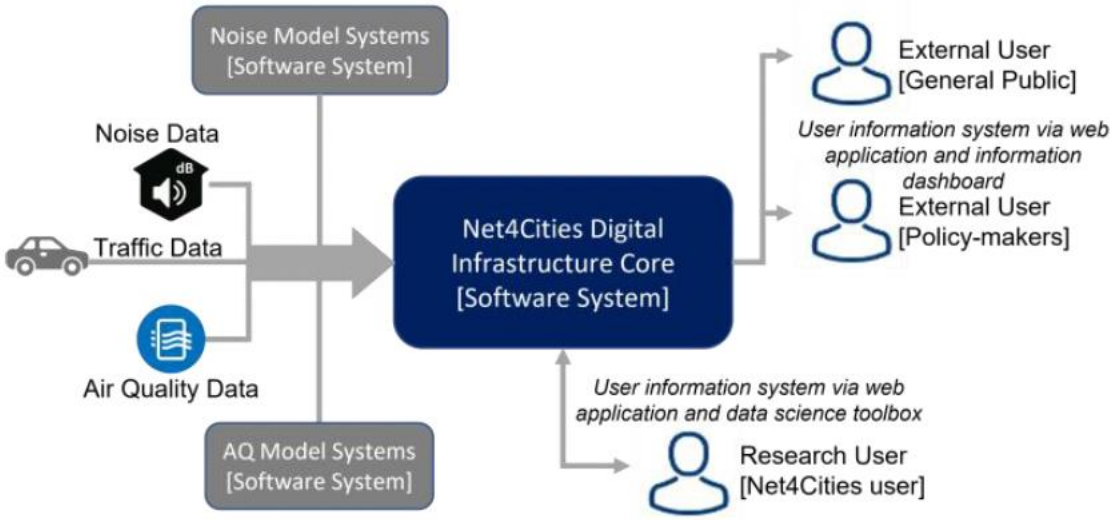


Figure 1: Net4Cities digital infrastructure concept diagram.

In the overview of requirements, we have emphasized two areas: data from measurements and data from models. These categories consist of subdivisions related to air pollution data, noise data and traffic data. The focus is to ensure quality control and integrating APIs for machine to machine (M2M) data transmission.

2.1 Measurement Data

2.1.1 Air Pollution measurement data

- Instrument Observation: Instruments are set up to monitor both (regulated) air pollutants and greenhouse gases, such as particulate matter (PM), nitrous oxide (N_2O), methane (CH_4), carbon dioxide (CO_2) and volatile organic compounds (VOCs), and emerging pollutants[2] such as particle number concentration (PNC), Lung Deposited Surface Area (LDSA), ammonia (NH_3), and black carbon (BC). These instruments provide real-time data with high temporal resolution, or aggregated when the sampling technique does not allow such resolution (e.g., ammonia will be measured by passive sampling with monthly resolution).

- Metadata: It includes the location coordinates of the measurements, altitude details, instrument identification numbers and specifics on measurement and sampling techniques.
- Quality control: A procedure to identify and correct data anomalies. This includes comparing data from neighboring instruments adjusting for calibration drifts and detecting any outliers.
- Data storage solutions to enable retrieval based on time stamps, location or pollution levels. This ensures access for real-time monitoring and historical analysis purposes.

2.1.2 Noise Measurement Data

- General Noise Levels: Data, from an indicator register that captures noise levels across the spectrum (usually 20 Hz to 20 kHz)[3]. This information is important for understanding noise pollution levels in general.
- Frequency Distribution Data: Information from the spectrum register that shows noise levels across frequency bands. This breakdown is crucial for identifying sources of noise and their effects.
- Sound Level: Similar to the General Noise Levels and Frequency Distribution Data but focusing on levels that vary based on the Sound Level Meter (SLM) setup. Percentiles such as L90, L50, L10 describe the noise level exceeded for 90%, 50% and 10% of the measurement period respectively. These values are useful for understanding how noise levels vary with high temporal resolution.
- Metadata: Includes location coordinates of measurements, altitude, instrument ID and details of Sound Level Meter (SLM) configuration/setup data. This involves specifics about microphone sensitivity, frequency weighting (e.g., A,C,Z-weighting) and time weighting (fast slow, impulse), during measurements.
- Ancillary and Compliance Data (markers): Records of time points highlighted as important during the measurement process, which could signal occurrences or disturbances needing additional quality control.

Given the need for efficient data storage, we propose storing noise level data at 1-minute intervals. This method provides the granularity necessary for detailed analysis. Longer-term indicators, such as hourly averages, can then be derived from these 1-minute values as required. This approach ensures optimal use of storage by maintaining detailed data only at the smallest necessary interval. Alternatively, we could consider storing longer-term aggregates directly, such as 1-hour averages, which might reduce the complexity of on-demand calculations but at the cost of losing finer granularity. We need to evaluate whether it is more efficient to store only 1-minute interval data and calculate longer-term indicators as needed, or if it's better to store these longer intervals directly for easier but less detailed access.

2.1.3 Traffic Counting Data

- Vehicle Numbers: The count of cars passing by a given location, on a road during a period. This is crucial for estimating traffic emissions and noise pollution levels.
- Vehicle Categorization: Different types of vehicles release varying amounts of emission amount and pollutant and produce different levels of noise. Data should distinguish between vehicle categories like motorcycles, cars, vans, buses and trucks.
- Traffic Patterns: Details about the speed and volume of traffic. Slow-moving or idling traffic can lead to higher emissions per vehicle.

- Time Sensitive Information: Data specific to time periods, such as daily and seasonal variations in traffic flow. This helps in understanding peak traffic times and off-peak times, which have implications for emission noise pollution levels.
- Geographic Details: Location-based data where traffic counts are done. This assists in identifying areas with traffic that may contribute more to pollution levels.
- Road Slopes and Design: Information on road elevations, declines and layout (e.g., intersections, roundabouts).

2.2 Model Data

The concept diagram (see Figure 2) illustrates the needed data to be stored in the Net4Cities Data Hub to conduct an air quality dispersion simulation, including improved source apportionment, noise simulations and the storage of the output from these models.

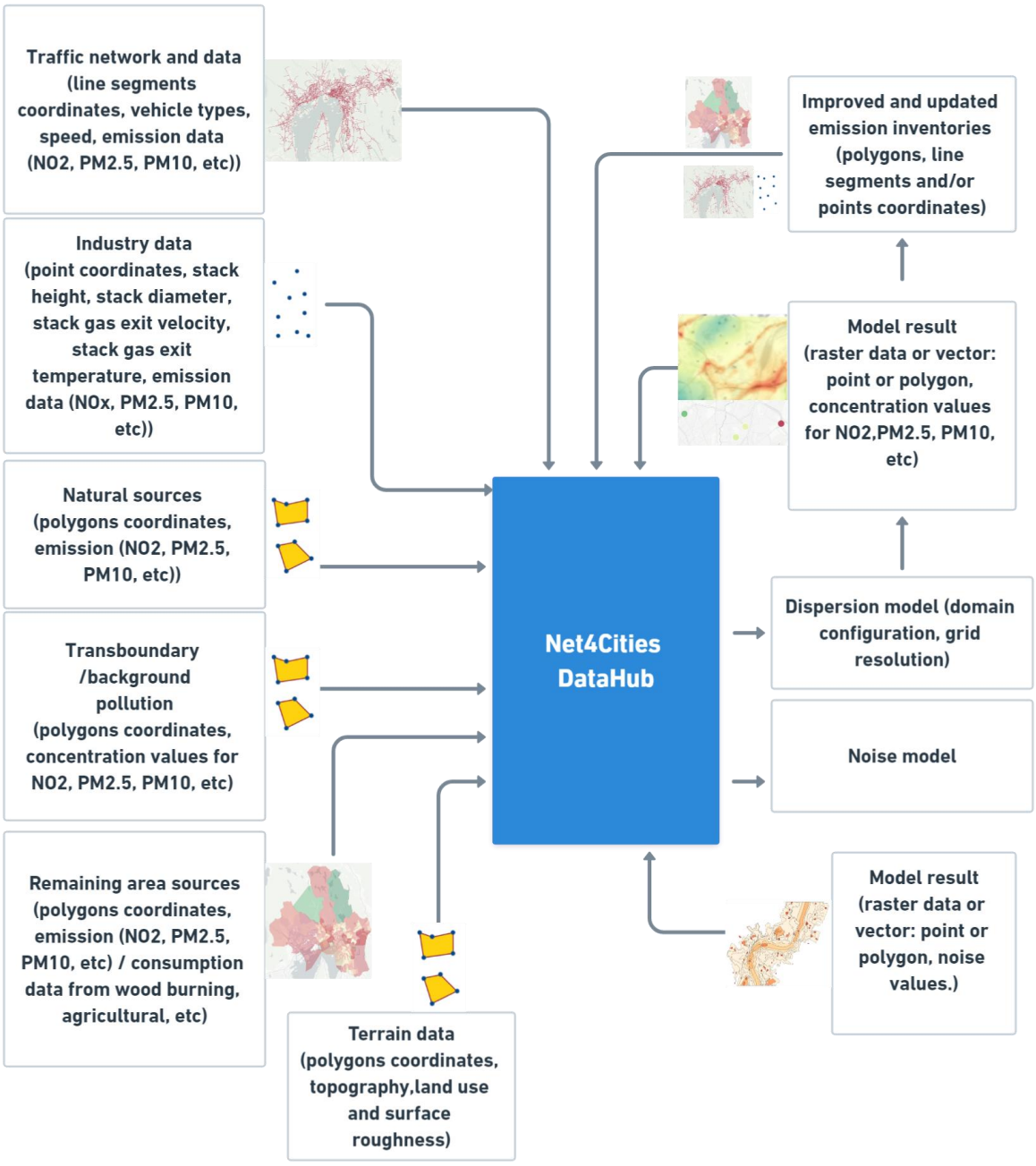


Figure 2: Net4Cities Data Hub conceptual diagram.

2.2.1 Air Pollution Modelling Data

Air quality dispersion modelling[4] is used to calculate how air emissions are dispersed in the atmosphere over time and space. This type of modelling is essential for understanding the impact of different pollutants from various sources, such as industrial plants, traffic, house heating, agricultural activities, etc., on the air quality of surrounding environments[4].

There are several important data that are required to carry-out air quality dispersion modelling scenarios. Within the Net4Cities Data Hub, we will store traffic-related emissions data as line sources and industrial data as point sources. Natural emissions and other remaining sources that don't fit into the line or point categories are classified as area sources (e.g., as gridded data). Additionally, we will include transboundary/background sources and the necessary terrain data. Meteorology data as wind speed and direction, temperature, humidity, and atmospheric stability play a critical role in determining how pollutants travel and dilute over time. But due to the complexity and volume of such data, these will probably not be stored in the platform, only linked. Users are encouraged to obtain meteorological data from reliable sources to ensure the accuracy and effectiveness of the air quality dispersion modeling.

Line sources: Line sources are critical for modelling urban air quality because they simulate the continuous release of pollutants from traffic, which is often a major contributor to local air pollution levels. The inputs required include:

- Coordinates of the traffic network as line paths (series of coordinates defining the path).
- Road slopes.
- Vehicle Speeds.
- Traffic composition (i.e. vehicle types/categories).
- Traffic counts (annual average daily traffic).
- Emission Factors for different pollutants (NO_x, PM_{10}, $PM_{2.5}$, SO_2, etc.) and vehicle types (EURO, heavy, etc.).
- For particles (PM) one also needs precipitation data for resuspension.
- Traffic variation (weekly- and diurnal variations).

Point Sources: These are typically single, identifiable sources of emissions that release pollutants from a fixed location and known height, such as smokestacks or exhaust vents. The inputs required include:

- Location: latitude, longitude, and elevation.
- Height of the release point above ground level.
- Exit temperature of the emissions which affects plume rise.
- Exit velocity of the emissions which impacts plume behavior.
- Pollutant type (NO_x, PM_{10}, $PM_{2.5}$, SO_2, etc.) and emission rate (e.g., grams per second).
- Temporal profiles in emissions (weekly- and diurnal variations).

Area Sources: These sources emit pollutants from a defined area rather than a single point. Examples include residential heating, agricultural fields, landfills, or small industrial sites. Inputs needed are:

- Coordinates of the area(s) defined as a grid(s).
- Pollutant type (NO_x, PM_{10}, $PM_{2.5}$, SO_2, etc.) and emission rate (e.g., tones per year).
- Temporal profiles in emissions (weekly and diurnal variations).

Boundary Conditions (Transboundary Concentrations): When modeling air quality, boundary conditions are critical for the model. These boundary conditions, also known as transboundary concentrations, are the air masses coming into the model domain. The inputs required include:

- Hourly concentration values of the pollutant to be modelled entering the model domain. For non-inert pollutants like NO_2, one also needs O_3, NO additional to NO_2.

Terrain Data: Terrain data is critical for air quality dispersion modeling because it significantly influences how air pollutants disperse in the environment[8]. Here are the key aspects of terrain data that are needed for effective modeling. The inputs required include:

- Coordinates of the grid cells defined as a square (in the data hub this will defined as polygons).
- Terrain value for each grid cell.
- Terrain type (elevation data, Land Use and Land Cover (LULC) data, surface roughness, vegetation types, etc.).

In addition to these data, air quality dispersion models require several 2- and 3-dimensional meteorological parameters such as wind speed and wind direction, temperature, atmospheric stability, mixing height, relative humidity, precipitation, etc. We have decided not to store meteorological data in the Net4Cities Data Hub due to the challenges associated with storing such data and the unclear purpose of storing all this information at this stage. Therefore, meteorological data will not be described further in this architecture document. It may be useful to further discuss the possibility of using terrain data and meteorological data, and to consider their interrelated roles in diffusion models; however, that is not in the scope of this document.

2.2.2 Noise Modelling Data

There are several models to evaluate, through calculation methods, the environmental noise levels produced by the main noise sources (traffic, railway, industry, airports) existing in a zone. The European Directive 2002/49/EC states that the method that must be used in Europe is CNOSSOS-EU. Proper noise modeling requires a comprehensive set of data starting with source data, which includes the type, location, and power levels of the noise source. This data should detail the sound power levels in decibels and the operating schedules, indicating frequency and operational times, as these factors significantly influence noise exposure.

In the case of traffic noise, the sound power levels are obtained from the traffic data, specifically from the amount of each vehicle class (passengers, light trucks, heavy trucks and motorbikes) and their respective speeds, along with the characteristics of the road (type of pavement, etc). Receiver data is also crucial, encompassing the specific locations and heights at which noise impacts are assessed, such as in residential areas, schools, or hospitals. This ensures that evaluations are focused on areas where noise may have the most effect. Terrain and obstacle data are vital for modelling how sound travels through different environments. This includes the digital elevation model, and the buildings and barriers that may influence the sound propagation path, from the source to the receptor.

We will not store all this data in the Net4Cities Data Hub, instead, we will focus on line sources (traffic) and terrain data.

Line sources: Understanding how noise disperses from line sources, such as road traffic, is crucial for accurate noise modeling in urban environments and the design of effective noise control measures. The inputs required include:

- Coordinates of the traffic network as line paths to map the exact locations of roads, highways, and other transportation routes where noise generation occurs.
- Vehicle speeds at which vehicles travel along these paths directly influence the sound power levels generated.

- Traffic composition (i.e., vehicle types/categories) has distinct noise emission characteristics, contributing differently to the overall noise level.
- Traffic counts (annual average daily traffic) quantify the number of vehicles passing through a specific section of the traffic network. Higher traffic volumes generally lead to increased noise levels, making this a critical factor in noise modeling.
- Traffic variation (weekly- and diurnal variations) can vary significantly depending on the time of day and day of the week. It is important to account for these variations, as noise levels can fluctuate based on peak and off-peak traffic periods.
- The type of pavement can affect the noise generated by vehicles. They have varying levels of noise absorption and reflection capabilities, influencing the overall sound power levels emitted by traffic.

Terrain Data: Terrain data is crucial for noise modeling because it significantly influences how sound waves travel and interact with the environment. The inputs required includes:

- Coordinates of the grid cells defined as a square (in the data hub this will defined as polygons).
- Terrain value for each grid cell.
- Terrain type (elevation data, LULC data, surface roughness, vegetation types, etc.).
- Obstacles (buildings and barriers/walls in the surrounding of the NMT).

3. Quality Goals

When creating a data hub that manages huge amounts of temporal and spatial data, it is essential to establish clear quality objectives. The primary goals for ensuring quality in the Net4Cities Data Hub are robustness, efficiency, scalability and the ability to provide insights.

The key quality goals for the Net4Cities Data Hub are:

Performance and Efficiency

- Low Latency Processing: Aim for minimum delays in processing and querying data even for handling extensive spatial and temporal datasets.
- Optimized Data Storage: Utilize HBase's efficient storage mechanisms and GeoMesa's indexing strategies to optimize data storage, reducing storage costs and speeding up query responses[5].

Scalability and Flexibility

- Horizontal Scalability: Develop a system that can scale horizontally by utilizing Apache Sparks' distributed computing capabilities and Kafka's scalable messaging system to manage growing data volumes and concurrent users effectively[6].
- Adaptive Data Models: Support data models accommodating various datasets with different spatial and temporal resolutions.

Data Integrity and Accuracy

- High Data Quality: Implement checks to ensure quality, consistent data throughout the hub, particularly when integrating information from multiple sources.
- Temporal and Spatial Precision: Precision in representing spatial data should be maintained at a high level along with effective querying utilizing GeoMesa's features for spatio-temporal indexing [5].

Reliability and Availability

- High Availability: The data hub services must ensure availability through fault tolerance strategies and recovery plans utilizing Kafka's replication and Sparks' resilient distributed datasets (RDDs)[8].
- Data Recovery: Robust backup and recovery processes should be implemented to safeguard against data loss and ensure business operations.

Security and Compliance

- Data Security: Where needed, strict data security measures must be enforced, including encryption during transit and rest, access controls and audit logging to safeguard information[7].
- Regulatory Compliance: Adherence to data protection regulations and standards is crucial to ensure compliance with industry requirements[7].

Usability and Accessibility

- User-Friendly Interfaces: User-friendly interfaces should be provided for both technical experts and non-experts to enable querying, visualization and analysis of the data.
- Accessibility: Sufficient documentation on the architecture, data models, APIs, and usage examples of the system should be offered to enhance ease of use.

Monitoring and Maintenance

- Monitoring: Utilize monitoring software to keep track of the well-being of the system, allowing for detection and fixing of problems.
- Maintenance: Create a system that will be easy to maintain with instructions, for updates, expansion and optimizing performance.

3.1 Stakeholders

The list identifies the main stakeholders of the Net4Cities Data Hub as follows. Clarifying these roles ensures that expectations around implementation, system reliability, and data accessibility are clearly understood and met. This will facilitate better integration, compliance, and collaboration across all parties involved.

- Stakeholders, such as development teams, project coordinators, and external researchers need to understand the system architecture as they have diverse expectations and roles that significantly influence the structure and functionality of the Net4Cities Data Hub This clarity ensures all parties understand their influence on the system's design and functionality.
- Individuals or entities that require assurance of the system architecture's reliability.
- Collaborators involved in working with or deploying the system architecture.
- Parties needing access to the system's architectural documentation for their responsibilities.
- Key decision-makers responsible for guiding system decisions and development.

Table 1. Roles and responsibilities.

Role	Expectations
NILU's Development Team	Clear and measurable implementation plan and guidelines that outline the structure and functionality of the Net4Cities Data Hub
RIFS's Development Team	Clear and measurable implementation plan and guidelines that outline the structure and functionality of the Net4Cities Data Hub
Development Team using the APIs (for instance EarthSense)	Clear onboarding process with sufficient documentation (including code examples) that covers all aspects of the APIs. Stability and reliability of the APIs, including proper versions. Support and community engagement, including a Community Forum where users/developers can ask questions.
Project Coordinator	Integrate inputs from relevant work packages and partners, conduct successful piloting activities, and finally deliver a high-quality system that supports the unique value proposition.
External Researchers	Expect the system to provide high-quality, accessible data with advanced analytical tools, ensure data accuracy and security, support integration with other research tools, and facilitate collaboration and data sharing.
Local and national environmental agencies	Expect the system to accurately collect and report data, ensure data accessibility and security, support regulatory compliance, integrate with other systems, promote public transparency, and be user-friendly and sustainable.
Relevant project partners	Expect the system to enable seamless collaboration, provide reliable and accessible data, support integration with their existing tools, ensure data security, and facilitate efficient project management and decision-making.
The European Commission	Delivery of high-quality systems to ensure long-lasting impacts by providing reliable and robust performance, fostering sustainable practices, and supporting ongoing innovation.

4. Architecture Constraints

The constraints of the Net4Cities Data Hub will be reflected in the final version. This section shows them and if applicable, their motivation. In addition to these constraints, one must adhere to the Net4Cities Data Management Plan (https://syncandshare.desy.de/index.php/s/NHCMozA5zECAGjX).

4.1 Technical Constraints

This table outlines critical technical constraints for building, operating, and maintaining a robust platform for handling instruments and modelled spatial-/temporal data using Apache technologies, Scala, Java, and C#. Each constraint is motivated by the need to ensure efficient, scalable, and reliable data processing and management.

Table 2. List of Technical Constraints (TC).

	Constraint type	Constraint Description	Background / motivation
Software and programming constraints			
TC1	Language Interoperability	Ensure smooth interoperability between Scala, Java, and C# components.	Utilizing the strengths of each programming language to facilitate communication and integration, between components.
TC2	Real-Time Data Processing	Implement real-time data processing pipelines using Apache Kafka and Apache Spark.	Analyzing real-time air quality and noise sensor data to offer insights and actionable steps.
TC3	Spatial Data Handling	Efficiently manage and query spatial data using GeoMesa and GeoServer.	Specialized handling and querying capabilities are crucial for data to support information systems (GIS) and spatial analytics.
TC4	Data Serialization	Use Apache Avro for data serialization to ensure efficient and compact storage and transmission of data.	Avro presents a compact and swift serialization format for managing high-volume data and transmitting it over networks effectively.
TC5	Concurrency Management	Utilize concurrency frameworks and libraries in Scala, Java, and C# to handle multiple data streams concurrently.	Efficiently managing data streams to ensure processing leveraging the concurrency features unique to each programming language.
TC6	Data Ingestion	Develop robust data ingestion pipelines using Apache Kafka and custom ingestion scripts in Scala/Java/C#.	Ensuring reliable and scalable ingestion of large volumes of instrument and model data from air quality instruments and model inventories (emission, terrain data etc.) and modelled air quality concentrations and noise decibels results.
TC7	Fault Tolerance	Utilize HBase's built-in support for data replication and region server failover to ensure continuous availability.	Ensuring the system can recover from failures without data loss.
TC8	Schema Evolution	Manage schema changes over time using Schema Server and Avro's schema evolution features.	Supporting the evolution of data models without causing disruptions in existing applications while ensuring both forward and backward compatibility of data structures.

TC9	API Development	Develop RESTful APIs for data access and manipulation using frameworks in Java and C#.	Offering interfaces that are user friendly for accessing data and integrating with other systems seamlessly.
TC10	Data Storage	Store spatial and temporal data in Hbase and meta data in PostgreSQL, ensuring optimized storage for different data types.	Using appropriate storage solutions for different types of data to ensure efficient storage, retrieval, and query performance
TC11	Data Querying	Implement complex querying capabilities using Apache Phoenix and GeoMesa.	Enabling advanced querying of both time-series and spatial data to support detailed analysis and reporting.
TC12	Performance Optimization	Optimize performance through JVM tuning for Scala/Java applications and .NET optimizations for C# applications.	Ensuring the applications run efficiently and handle large-scale data processing without performance degradation.
TC13	Security and Access Control	Implement robust security measures, including encryption, authentication, and authorization.	Protecting data from unauthorized access and ensuring compliance with security standards and regulations.
TC14	Scalability	Design the system to scale horizontally using distributed processing frameworks like Apache Spark.	Ensuring the platform can handle increasing data volumes and user loads by scaling out across multiple nodes.
TC15	Monitoring and Logging	Implement comprehensive monitoring and logging using tools compatible with Scala, Java, and C#.	Providing visibility into the system's operations and performance, facilitating troubleshooting and performance tuning.
Operating System Constraints			
TC16	Deployable	Deployable to Linux server.	The application should be deployable through standard means on a Linux-based server.
TC17	Patch Management	Regularly update the operating system to apply security patches and updates.	Ensure the system remains secure and up to date with the latest protections and performance improvements.

TC18	System Monitoring and Logging	Use OS-specific tools for logging.	Provide insights into system performance and facilitate troubleshooting by capturing detailed operational data.
Hardware constraints			
TC19	Processing Power	Ensure sufficient CPU cores and processing power to handle high computational workloads.	Support intensive data processing tasks and real-time analytics without performance degradation.
TC20	Memory	Provide adequate RAM to handle large datasets and support in-memory processing.	Enable efficient data processing and reduce the need for disk I/O, enhancing performance.
TC21	Storage Capacity	Ensure ample storage capacity with fast I/O, using SSDs for critical data operations.	Accommodate large volumes of instrument and model data, ensuring quick data access and retrieval.
TC22	Network Bandwidth	Ensure high network bandwidth and low latency to support real-time data transmission.	Facilitate efficient data ingestion, processing, and communication between distributed components.

4.2 Organizational Constraints

Table 3. List of Organizational Constraints (OC).

	Constraint type	Constraint Description	Background / motivation
OC1	Budget	Ensure the project stays within the allocated budget for development, deployment, and maintenance.	Control costs and ensure the financial feasibility and sustainability of the project.
OC2	Staffing	Ensure availability of skilled personnel for development, deployment, and support.	Ensure the project has the necessary human resources with appropriate skills and expertise.
OC3	Training	Provide adequate training for staff in modern technologies and processes.	Equip team members with the knowledge and skills needed to effectively use and support the platform.
OC4	Stakeholder Alignment	Ensure alignment with stakeholder expectations and requirements.	Maintain clear communication and agreement with stakeholders to ensure project goals are met.

OC5	Change Management	Implement robust change management processes to handle updates and new requirements.	Ensure smooth transitions and minimize disruptions when changes are made to the system or processes.
OC6	Governance	Implement proper governance structures to oversee project progress, project risk and decision-making.	Ensure accountability, transparency, and strategic alignment throughout the project lifecycle.

4.3 Legal Constraints

Table 4. List of Legal Constraints (LC).

	Constraint type	Constraint Description	Background / motivation
LC1	Data Privacy	Check all data processing steps for general data protection regulation (GDPR) compliance.	Ensure that all legitimate interests of the actors involved regarding data privacy and security are met.
LC 2	Licenses	Check all hardware and software components involved about their respective license models.	Ensure license-compliant operation of the entire system, considering the available budget, if necessary, even beyond the project term.
LC3	Intellectual Property Rights	Consideration of the interests of project partners from the private sector with regard to non-freely available software components.	Ensuring the legally secure operation of such components in the project context.

5. System Scope and Context

This chapter describes the environment and context of the Net4Cities Data Hub, in other words, who will use the system, and which other systems Net4Cities Data Hub depends on.

5.1 Business Context

Error! Reference source not found. illustrates the Net4Cities Data Hub in a business context.

The Net4Cities Data Hub aims to combine data sources and analytical models to offer insight into urban environmental conditions. This integration supports decision-making and boosts public awareness by providing targeted insights, for researchers, policymakers and the public.

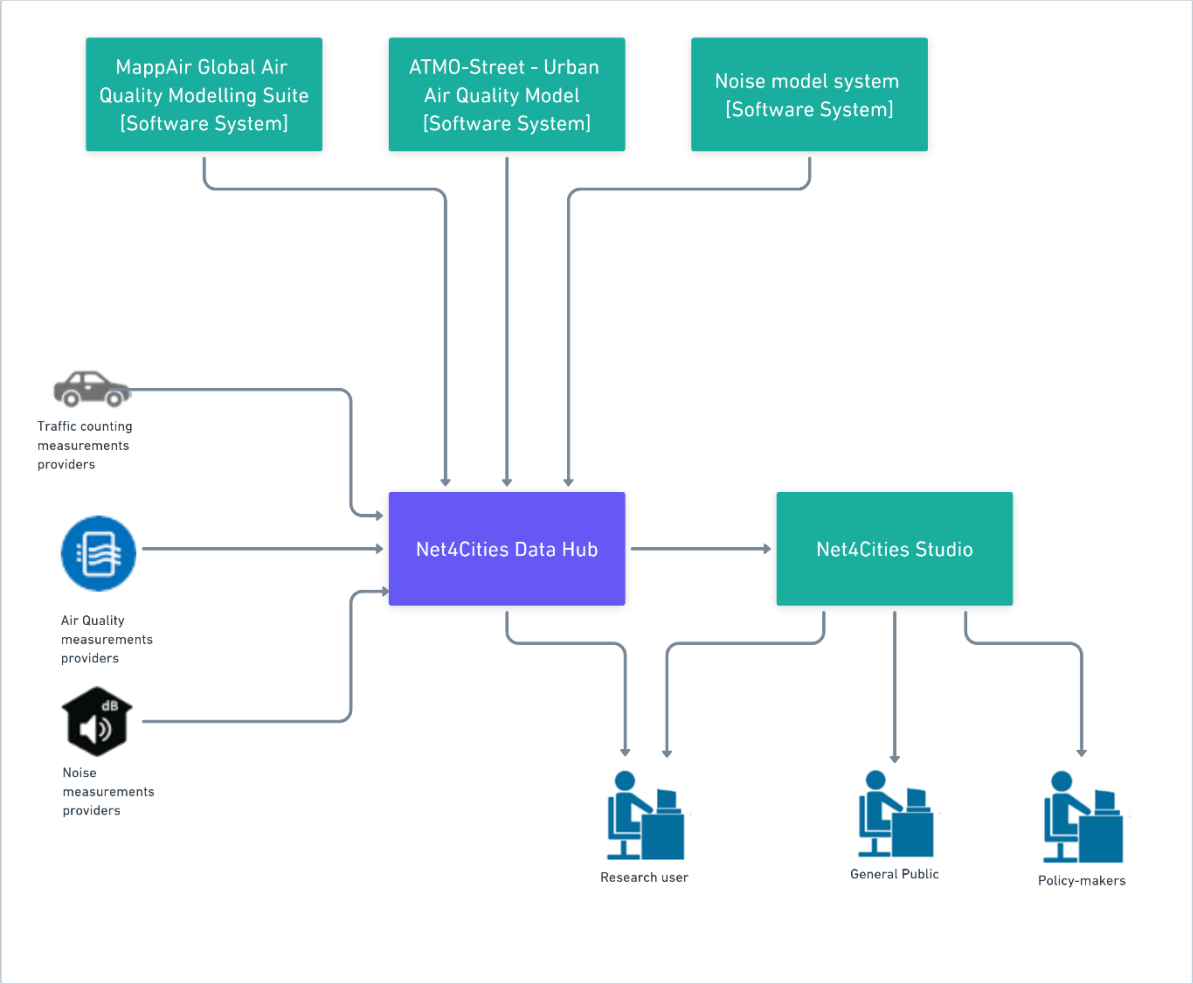


Figure 3. Net4Cities Data Hub in a business context.

Key Components:

1. Data Sources:

- MappAir® is Earth Sense's proprietary modeling engine designed to create air pollutant dispersion models and maps for key pollutants. It can process diverse pollution sources and track the dispersion and evolution of emissions. MappAir® handles various spatial scales and temporal resolutions, offering real-time (hourly) source apportionment of PM2.5 and NOx at a spatial resolution of up to 10 meters.
- ATMO-Street - Urban Air Quality Model: Focuses on city air quality data providing in-depth insights into conditions at a local level.
- Noise Model System: Provides information on noise pollution levels for assessing sustainability and health impacts.
- Traffic data: Generates emissions and noise data from vehicle sources serving as input, for air quality and noise models.
- Air Quality Monitoring Stations: Provide real-time air quality data to ensure the system stays updated with the most recent environmental conditions.

2. Net4Cities Studio:

- Researcher User: This platform offers advanced analytical tools and detailed data displays to assist in-depth environmental research.
- General Public: This interface is designed inform citizens about their local environmental conditions by providing information and data visualizations, thereby promoting public awareness.
- Policy Makers: This platform provides insights and visualizations to aid evidence-based decision making to support the design of environmental policies.

By integrating these components, the Net4Cities system, which includes the Net4Cities data hub and Net4Cities Studio aims to create a platform that enhances understanding of urban environmental conditions and provides valuable insights for various stakeholders, thereby contributing to the development of smarter and more sustainable cities.

6. Technical Context

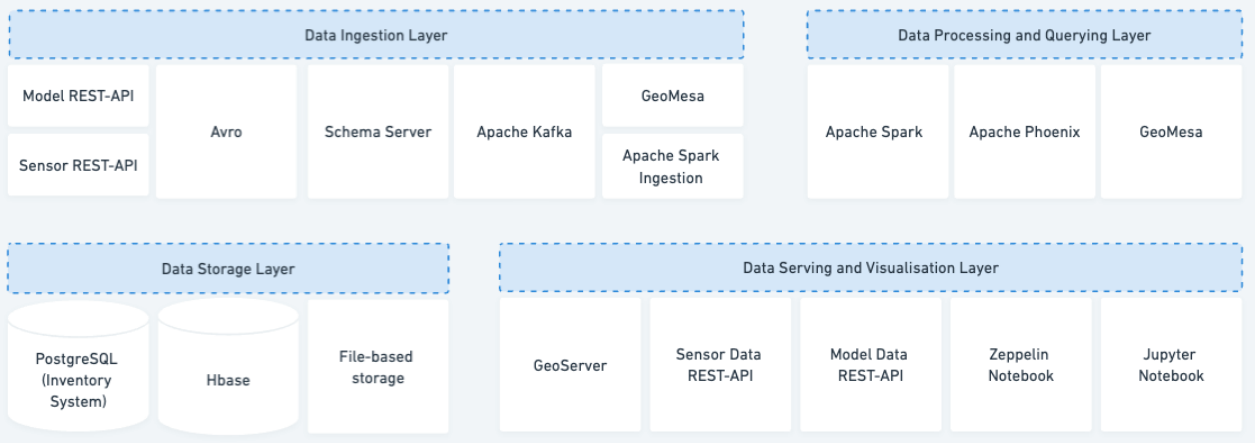


Figure 4. Net4Cities Data Hub in a technical context.

The diagram above illustrates the Net4Cities Data Hub in a setting presented as a layered structure that outlines components and their interactions across four key layers: Data Ingestion, Data Storage, Data Processing and Querying, and Data Serving and Visualization. Here is an overview of the interfaces and how specific domain inputs/outputs are linked to these channels.

6.1. Data Ingestion Layer

This layer is responsible for gathering and importing data from sources into the system.

- **Instrument REST API:** This API interface manages data ingestion from instruments, including devices, environmental instruments and other hardware that collects real-time data.
- **Model REST API:** This API interface focuses on importing data related to models. It can include inputs, outputs, parameters and metadata of machine learning models.
- **Avro**[8] is a data serialization system utilizing JSON to define data schemas. It facilitates data exchange by allowing serialization/deserialization while ensuring schema compatibility over time. This feature proves beneficial in maintaining compatibility across versions of data.
- The **Schema Server** is responsible for managing data format schemas to maintain data structures ensuring data integrity and compatibility across different systems.

- **Apache Kafka** plays a role as a distributed streaming platform for real-time data ingestion supporting throughput and low latency data streaming. It facilitates the collection and transfer of volumes of data across system components.
- **Apache Spark** serves as an engine for data ingestion, capable of handling both real-time and batch-processing tasks making it suitable for managing large scale datasets.
- **GeoMesa** is a spatial data management system that works in conjunction with Apache Kafka to ingest information. It excels in executing queries that involve temporal dimensions.

Input/Output Mapping:

- Model data flows through the Model REST API before being serialized using Avro to optimize storage efficiency and processing speed.
- Data, from instruments is received via the Instrument REST API, managed by the Schema Server and then sent through Apache Kafka for processing.
- Spatial and temporal data is brought in through GeoMesa, which might also utilize Apache Kafka for real-time geospatial data streaming.

6.2. Data Storage Layer

This layer holds the received data, for processing and retrieval.

- **PostgreSQL (Inventory System):** A database used to store structured inventory data. It ensures data reliability through ACID properties (Atomicity, Consistency, Isolation, Durability).
- **HBase:** A non-relational distributed database built to manage large scale datasets. It's well suited for storing time series data and other large datasets requiring read/write operations.
- **File-based storage**: This method is employed to store file types that do not align with database structures like large binary files, images, logs and raw data files.

Input/Output Mapping:

- Inventory data from PostgreSQL.
- Large-scale time series and vector data stored in HBase.
- Raster data types stored in a file-based system.

6.3. Data Processing and Querying Layer

This layer is responsible for handling the stored data and supporting querying processes. It utilizes the following tools for data processing:

- **Apache Spark**: A framework designed for quick processing of large datasets and distributing tasks across multiple computers.
- **Apache Phoenix**: Acts as a SQL layer on top of HBase enabling SQL based querying since HBase lacks SQL support. Phoenix translates HBase scan operations into SQL commands.
- **GeoMesa**: Enhances the system by providing spatial query capabilities allowing querying and processing of large geospatial datasets by integrating with Apache Spark and Kafka for real-time analytics.

Input/Output Mapping:

- Data from HBase is queried using Apache Phoenix.
- Large datasets processed with Apache Spark.

- Spatio-temporal data queried and processed with GeoMesa, facilitating complex geo-spatial analyses.

6.4. Data Serving and Visualization Layer

This layer provides interfaces for accessing processed data and tools for visualizing it.

- **GeoServer**: Serves geospatial data via standard web services (e.g., WMS, WFS). It allows for sharing and visualizing geospatial data over the web, supporting a wide range of geospatial data formats.
- **Instrument Data REST-API**: This API allows access to both processed and raw instrument data. It provides endpoints for querying instrument data based on various parameters.
- **Model Data REST-API**: Like the Instrument Data API, this interface allows access to model-related data, providing endpoints for querying processed and raw model data.
- **Zeppelin Notebook** and **Jupyter Notebook**: These interactive web-based notebooks are used for data exploration and visualization[15]. They allow data scientists and analysts to write and execute code in real-time, visualize data, and share insights.

Input/Output Mapping:

- GeoServer serves processed geospatial data.
- Instrument and model data can be accessed via respective REST-APIs.
- Data scientists and analysts can use Zeppelin and/or Jupyter Notebooks for interactive data analysis and visualization.

The system will efficiently process big amounts of diverse data types through well-designed pipelines. Each layer utilizes tools and frameworks to maintain data integrity, scalability and accessibility. This enables querying real-time processing, as well as interactive data analysis.

7. Solution Strategy

The design of the system is influenced by strategic tech choices a step-by-step breakdown strategy and a focus on achieving scalability, performance, adaptability and dependability using open-source technology widely supported and utilized by organizations within the community. Organizational choices like embracing methodologies and fostering skills will play a vital role in effectively developing and maintaining the system. This methodical approach will ensure an adaptable and scalable data processing system of managing various data formats and providing valuable insights through advanced processing and visualization tools.

Table 5. The requirements, architectural approaches, and description.

Goal/Requirement	Architectural Approach	Details
Scalability	Use of distributed systems	Apache Kafka for scalable data ingestion, HBase for scalable data storage, Apache Spark for distributed data processing.
Performance	Efficient processing and querying	Real-time data streaming with Apache Kafka, batch and stream processing with Apache Spark, fast SQL-based querying with Apache Phoenix.
Flexibility	Diverse storage and access methods, and support different programming frameworks	File-based storage for flexible data types, NoSQL (HBase) for large volumes of unstructured data, REST APIs for standardized data access. Support Scala, Java, Python, R and C#.
Consistency and Reliability	Data serialization and schema management	Avro for data serialization, Schema Server for consistent schema management, Apache Kafka for reliable data streaming.
Interactive Data Analysis	Use of collaborative tools	Zeppelin and Jupyter Notebooks for interactive data exploration and visualization.
Handling Spatio-Temporal Data	Specialized tools	GeoMesa for spatio-temporal data management and querying.
Structured Data Storage of inventory/metadata	Relational database	PostgreSQL for robust and complex query support.
Development Process	Agile methodologies	Iterative development, continuous improvement, and use of collaborative tools.
Data Privacy	State-of-the-art data security and access control	SSL-encrypted traffic, token- and/or password-based access to REST-API.

8. Building Block View

The perspective of building blocks presents a breakdown of the system into components, like modules, components, subsystems, interfaces, packages, libraries, frameworks, layers, partitions, levels, functions, macros, operations and data structures. It also includes their dependencies such as relationships and connections. This view is crucial for documenting the architecture as it offers a depiction of the system's structure and how its different parts interact.

- **Modules and Components:** These are self-contained units within the system that handle functions. For example, in the data ingestion layer, there are components like Model REST API, Instrument REST API, Avro, Schema Server, Apache Kafka, Apache Spark Ingestion and GeoMesa.
- **Subsystems**: These are sections of the system that are built up of components that work together. An example is the Data Storage Layer which comprises PostgreSQL, HBase, and file-based storage systems.
- **Interfaces**: They specify the functionality of components and how they interact with each other. For instance, the interfaces provided by REST API in the data provisioning and visualization layers.
- **Packages and Libraries**: Sets of classes and interfaces bundled together for reuse such as the libraries utilized by Apache Spark for handling data.
- **Frameworks**: Serve as platforms that support the structure of a system like how Apache Kafka is used for stream processing and Apache Phoenix is employed for querying HBase.
- **Layers and Tiers**: Represent varying levels of abstraction and separation of responsibilities within a system, including stages like data ingestion, storage, processing, querying and serving/visualization.

The building block perspective presents an arrangement of entities (black boxes) and detailed entities (white boxes) along with their explanations. This form of abstraction facilitates communication with stakeholders at a level without delving into technical specifics. Visualizing the architecture in this manner aids in managing, expanding and upkeeping the system efficiently.

The term "Black Box" refers to a higher-level component or subsystem that keeps the "under the hood" mechanism hidden like the data ingestion layer.

On the other hand, a "White Box" provides a view of the internal structure of the black box illustrating how specific components and modules interact and function within it. For instance, Apache Kafka, Schema Server and REST APIs collaborate within the data ingestion layer.

This hierarchical organization helps break down the system into parts. It ensures that each layer and component is understandable both on its own and in relation to the overall architecture. This document discusses details about the context, Level 1 and Level 2 diagrams.

The context diagram (shown in Figure 5) plays a role in system architecture by offering an overview of how a system engages with its external surroundings. It aids in defining boundaries and scope while fostering understanding among stakeholders regarding the system's operational environment and external dependencies. This clarity facilitates informed decision-making and effective communication, at every stage of system development and integration.

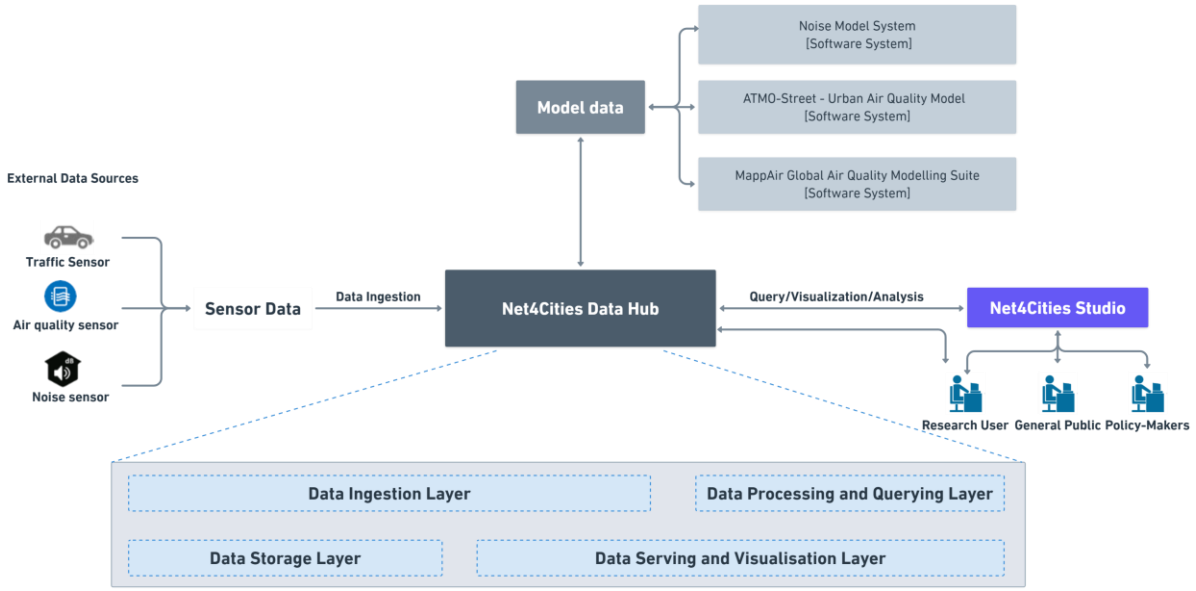


Figure 5. The context diagram.

Level 1 is a white-box description of the entire system and a black-box description of all contained building blocks, as shown in Figure 6. It serves as a central tool in system architecture by breaking down the system into its most important subsystems or components. It effectively highlights the important data flows and interactions between these components and illustrates how the major building blocks (black boxes) relate to each other. This mid-level view bridges the gap between the high-level context provided by the context diagram and the detailed design captured in the Level 2 diagram (Figure 7).

Level 2 zooms in on some of the Level 1 building blocks, as shown in Figure 7. Therefore, it contains white-box descriptions of selected Level 1 building blocks and black-box descriptions of the building blocks inside them. It becomes an important tool in system architecture by providing a detailed view of individual subsystems or components. It describes the internal structure, data flows, and complex interactions within each component.

The system is divided into layers. Building blocks to ensure it can scale be maintained efficiently and process data effectively. Each layer and its components serve purposes allowing the system to manage types of data in large quantities while maintaining performance and data integrity. This structured method supports development simplifies issue resolution and permits the adjustment of individual components as required.

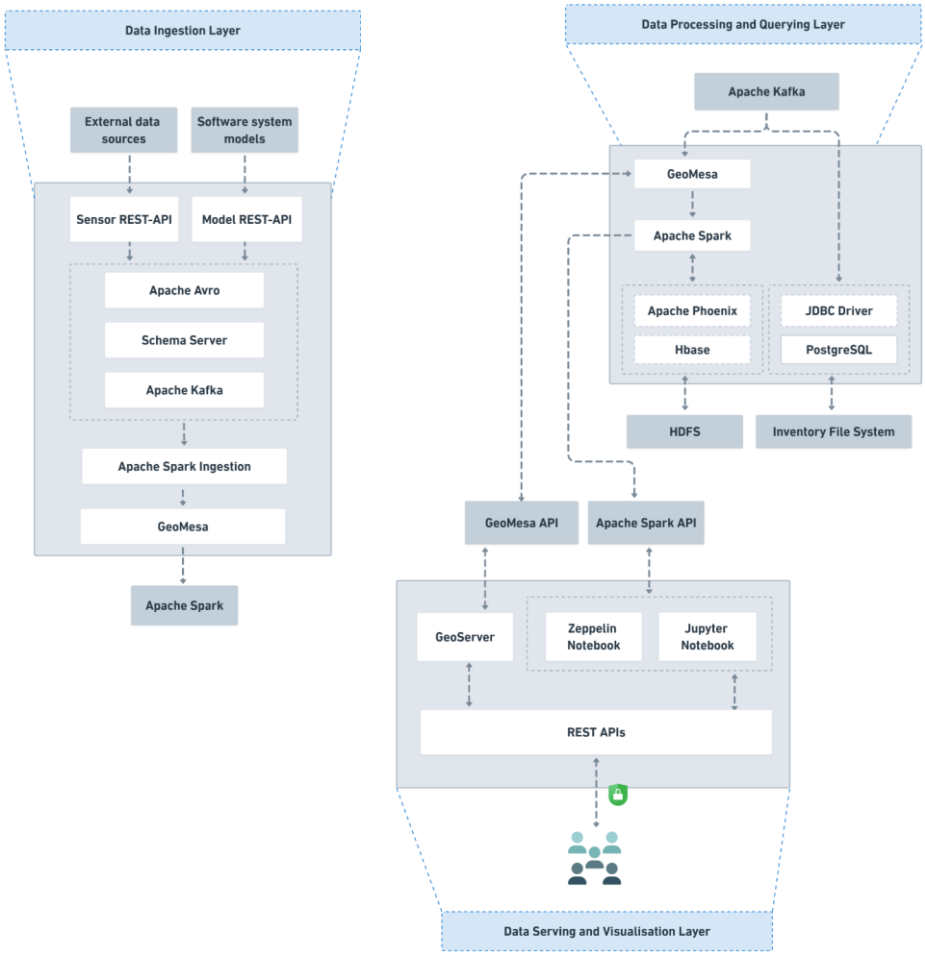


Figure 6. The Level 1 diagram.

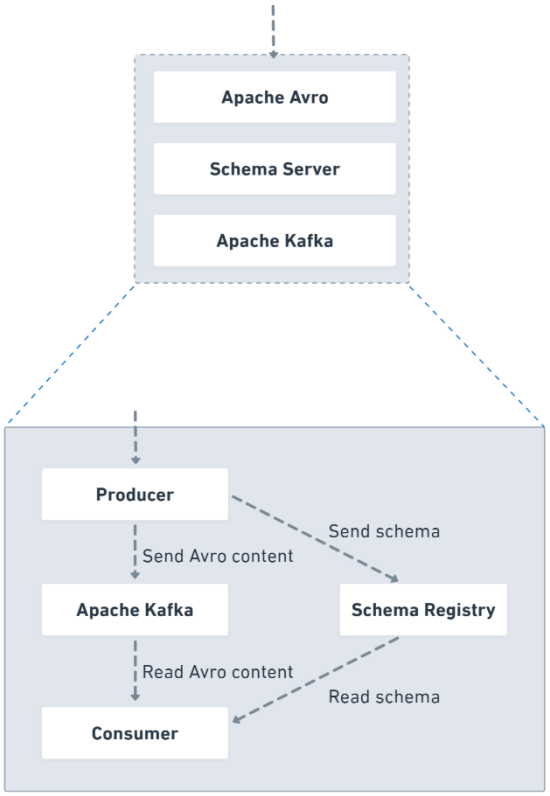


Figure 7. The Level 2 diagram.

8.1 Black Box Descriptions

The Model REST API handles fetching data related to models, into the system by accepting HTTP POST requests with JSON data and providing HTTP status codes and messages in response. While there may be concerns about scalability as data volume grows, the API prioritizes ensuring availability and minimal latency.

The Instrument REST API retrieves data from instruments into the system through HTTP POST requests containing instrument data in JSON format responding with HTTP status codes and messages. It focuses on real-time data processing. Fault tolerance acknowledges the challenge of managing instrument types.

Avro plays a role in serializing and deserializing data working with JSON-based data schemas to generate serialized binary information. It facilitates schema development and compact representation of data though managing schemas remains a concern. The schema server is responsible for storing and managing data schemas handling registration requests and retrieval responses to ensure consistency and speedy access to schemas while addressing conflicts in schema versions.

Apache Kafka drives real-time streaming and data ingestion processes by enabling producers to publish messages and consumers to subscribe to topics, for communication. The system offers data processing and minimal delays. However, it encounters difficulties in managing data streams and maintaining message sequences.

Apache Spark Ingestion deals with processing volumes of data in time and batch modes taking in data from diverse origins and generating refined data to a storage layer. It can scale up but handling resources efficiently and optimizing them might present challenges.

GeoMesa handles retrieval of spatial data and overseeing data streams and establishing indexed. It supports complex spatial queries despite the complexities involved in dealing with large scale geospatial data.

8.2 White Box Descriptions

Apache Spark executes data transformations, facilitates machine learning and graph processing and interfaces with data sources. Leveraging in-memory processing for computations Apache Spark offers APIs in Java, Scala, Python and R for data operations. Its fault tolerance features ensure scalability making it well suited for managing data processing tasks.

Structured components of Apache Spark Ingestion encompass elements such as Spark Streaming, Datasets and Data Frames. Spark Streaming handles the processing of data in time while Datasets and Data Frames are in charge of managing data. Spark SQL allows for running SQL queries on the data that is being ingested. These components work together with Kafka for real-time data processing, HBase for storage and Schema Server for validating schemas. Within Spark Streaming's structure there is a batch processing engine that ensures fault tolerance and seamless integration with Spark components. The micro batch engine processes data in batches fault tolerance is maintained through checkpoints to guarantee data reliability. It seamlessly integrates with datasets, data frames and external sources of data.

The internal architecture of GeoMesa includes features like indexes, integration with Kafka and powerful query engines. The spatial index efficiently manages geospatial data indexes to facilitate querying, while the Kafka integration allows for real-time ingestion of information while the query engine handles spatiotemporal queries. GeoMesa collaborates with HBase for storing data Apache Spark for processing tasks and GeoServer for serving up the processed information. The internal structure concerning indexes, within GeoMesa details how these indexes are created, managed effectively, and utilized. Index creation involves the development

of indexes, while index management includes tools for maintaining and updating these indexes. Using queries involves applying techniques to exploit indexes in query executions.

HBase is a distributed database tailored for handling datasets without relational structures. It stores data in columns and offers real-time read/write access across a distributed system like HDFS. HBase is built to expand horizontally by incorporating nodes to manage data volumes and workloads ensuring resilience and high availability. The database supports sharding and load balancing allowing execution of large-scale data operations. It is commonly used for storing time series data, instrument information and other substantial datasets that necessitate access and storage capabilities.

Apache Phoenix introduces an SQL interface on top of HBase that translates SQL queries into operations, on HBase providing SQL query functionalities. It facilitates joins and aggregations enabling users to conduct queries on HBase data. Phoenix enhances query efficiency using indexes facilitating retrieval and manipulation of data. It seamlessly integrates with existing HBase setups offering users a SQL interface to interact with the underlying HBase information.

PostgreSQL, a database system, is in charge of storing organized inventory information. It upholds data integrity by following ACID standards and facilitates queries and transactions. This system utilizes SQL for managing and retrieving data offering support for maintaining data integrity, indexing and handling transactions. With its availability and durability features, PostgreSQL is well suited for applications that demand uniform data storage.

File-based storage serves as a repository for various file formats providing flexibility in storing unstructured data. It accommodates file storage needs, offers accessibility, and management options making it ideal for housing unstructured data like documents, images and videos. File-based storage systems commonly integrate distributed file systems (such as HDFS) ensuring redundancy and fault tolerance. This element guarantees that data remains easily accessible, secure and well managed while supporting high throughput operations with integration across data processing systems. In terms of Data Observation/Visualization capabilities, GeoServer delivers information via web services using protocols like WMS and WFS. It empowers users to visualize and interact with data effectively while serving as a platform for managing and disseminating geospatial information. GeoServer offers support for data formats and seamlessly integrates with GeoMesa for advanced spatiotemporal queries.

The Instrument Data REST API serves as a user interface for accessing both processed and raw instrument data allowing flexible access to instrument information with a range of query options. It ensures the availability of data and enables real-time interaction. Similarly, the Model Data REST API provides endpoints to retrieve model inputs, outputs and metadata making it simple to access model data while facilitating integration with systems. Both APIs prioritize reliable data accessibility while ensuring availability and scalability.

Zeppelin Notebook and Jupyter Notebook are both web-based platforms designed for exploring and visualizing data. Users can write code, execute it, visualize the results, and share findings in a space that complies with data protection regulations. These notebooks support different programming languages and offer extensive visualization libraries to promote collaboration among data scientists and analysts. They are compatible with data sources and processing frameworks and can be used for conducting comprehensive data analysis and visualization tasks.

9. Runtime View

The system’s real-time perspective offers a view that focuses on how different parts interact and communicate while tasks are being carried out. During operations, the system goes

through stages such as data input, storage, processing, querying and delivery with multiple components collaborating to handle and respond to the data.

9.1 Data Ingestion Phase

In the data input phase, various sources feed information into the system. For instance, the model data and the input data for the model are sourced to be sent to the model REST API. It is serialized using Avro for data formatting and schema compatibility. Instrument data is received through the instrument REST API with schema management handled by the schema server to maintain data structures. Apache Kafka serves as a core component for real-time data streaming facilitating the transfer of instrument and model information across the system. GeoMesa manages data input and leverages Kafka for geospatial streaming. This phase ensures that all incoming data is correctly formatted, serialized and prepared for processing.

9.2 Data Storage Phase

Following ingestion, the processed information is directed to a storage solution. Structured inventory details (metadata) are stored in PostgreSQL—a database known for ensuring data integrity and supporting queries. Large amounts of time series and vector data are saved in HBase, a distributed database designed for handling datasets without relationships. Raster data and various file formats are stored in a file-based storage system indexed using PostGIS. This step ensures that all data is securely stored and can be easily processed and queried.

9.3 Data Processing and Querying Phase

In the phase of processing data, Apache Spark acts as an engine for managing extensive data processing tasks. Spark handles both batch and real-time data carrying out transformations and aggregations. For queries, Apache Phoenix adds a SQL layer on top of HBase to convert SQL queries into HBase operations enabling retrieval of data using SQL syntax. GeoMesa further enriches query capabilities by providing query functions for datasets. This phase guarantees processing of data for gaining insights through interrogation.

9.4 Data Serving and Visualization Phase

Data can be accessed and visualized via different interfaces and tools. GeoServer serves information through web services allowing users to visualize and interact with maps and spatial information. The Instrument Data REST API and Model Data REST API offer endpoints to access processed instrument and model data respectively. Interactive notebooks, like Zeppelin and Jupyter allow data scientists to analyze, visualize, write and run code in real-time. This phase ensures that end users can efficiently access, analyze and visualize data.

9.5 Sequence of Processes

In this system, several key scenarios can be visualized through sequence diagrams. Sequence diagrams illustrate the interactions between objects or components in a sequential manner and show the flow of messages and data over time (Figure 8).

9.5.1 Ingesting Instrument Data

In this scenario (refer to Figure 8), the instrument is the data source and is transmitting data to the system. The process commences when the instrument device sends an HTTP request to the instrument REST API with both the instrument data and its schema. Upon receiving this data, the instrument REST API promptly validates it against the schema provided by the schema server. If the instrument data does not adhere to the data schema, the schema server rejects it ensuring that only valid data is processed.

Following validation, the data undergoes serialization using Avro. An Avro dataset comprises instrument data and its schema creating a concise and efficient transmission format. This serialized information is then sent to an Apache Kafka topic for real-time streaming processing. Apache Kafka plays a role in this setup by streaming verified and serialized instrument data to consumers for further processing and analysis. This mechanism guarantees a dependable transfer of instrument data from producers to processing components within the system.

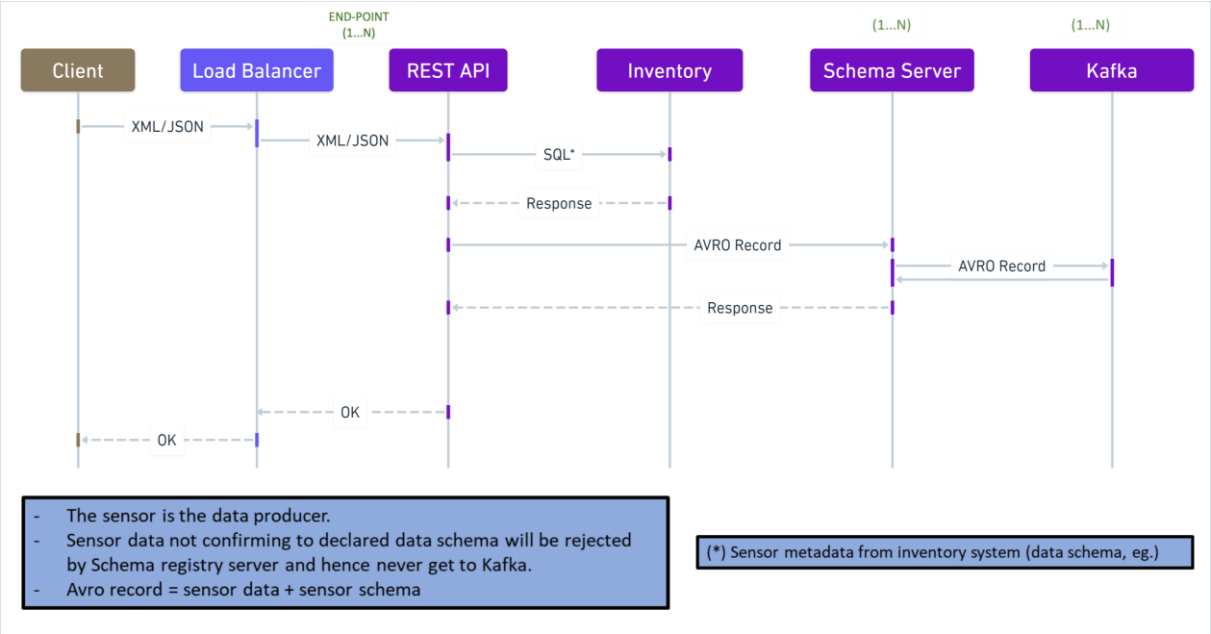


Figure 8. The sequence diagram for the Kafka producer for instrument data.

9.5.2 Ingesting Inventory Data and Storing

In this scenario (refer to Figure 9) the process of saving inventory information starts when the data is transmitted to the model REST API through an HTTP request. The model REST API verifies the inventory data to make sure it follows a format. Once confirmed, the data is transformed into Avro, which compacts both the data and its structure into a compressed form, for storage and transfer. The compacted inventory information is then saved in PostgreSQL. PostgreSQL organizes this information for retrieval and ensures its integrity. This series of steps guarantees that the inventory details are accurately captured, validated and stored in a manner that facilitates access and analysis.

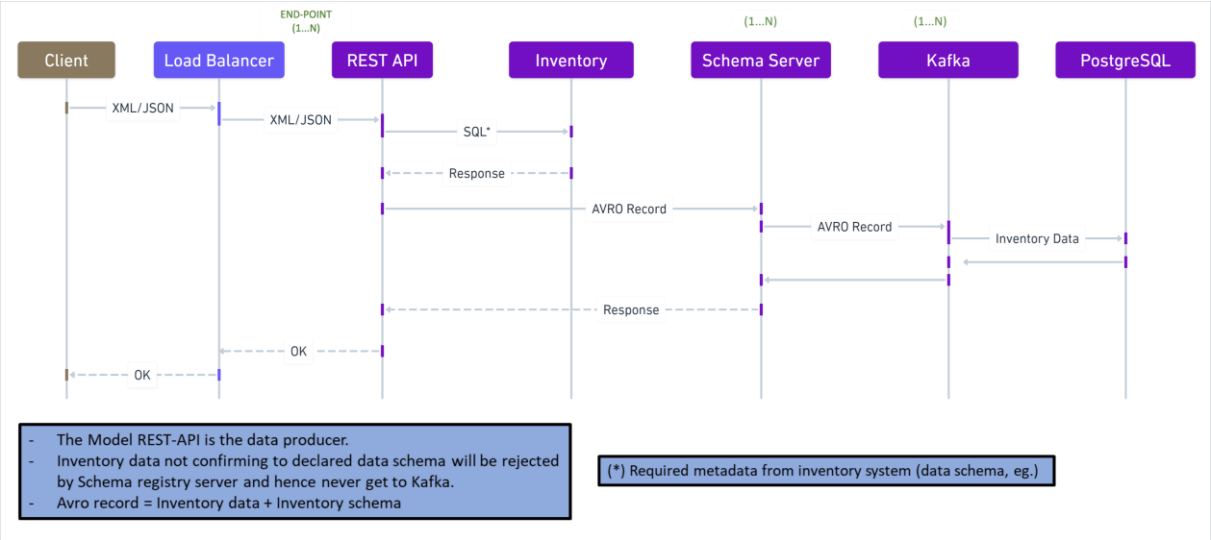


Figure 9. The sequence diagram for the Kafka producer for inventory data and persistence.

9.5.3 Persistence of Spatio-temporal Data and Quality Control

In this scenario illustrated in Figure 10 and Figure 11 Apache Kafka plays a role, within the system by transmitting validated and serialized instrument observation data to recipients for further analysis. This facilitates a dependable flow of instrument data from producers to processing components in the system. GeoMesa collaborates with Apache Kafka to intake data into Spark for processing. In this configuration the initial Spark Submit application functions as both a Kafka consumer and producer. It retrieves data from Kafka at intervals of 10 seconds, processes it, and stores the details in HBase. Instrument information requiring quality assurance is returned to Kafka for consumption by the Spark application, establishing a pipeline for processing. The second Spark Submit application operates as a Kafka consumer for quality control (QC). It also retrieves data periodically from Kafka every 10 seconds. Executes QC tasks such as applying QC flags on the data. Following processing, it updates the modified QC flags into HBase. This methodical approach ensures that data is not just captured and stored but is also meticulously reviewed and tagged with quality indicators thus upholding data integrity throughout the system.

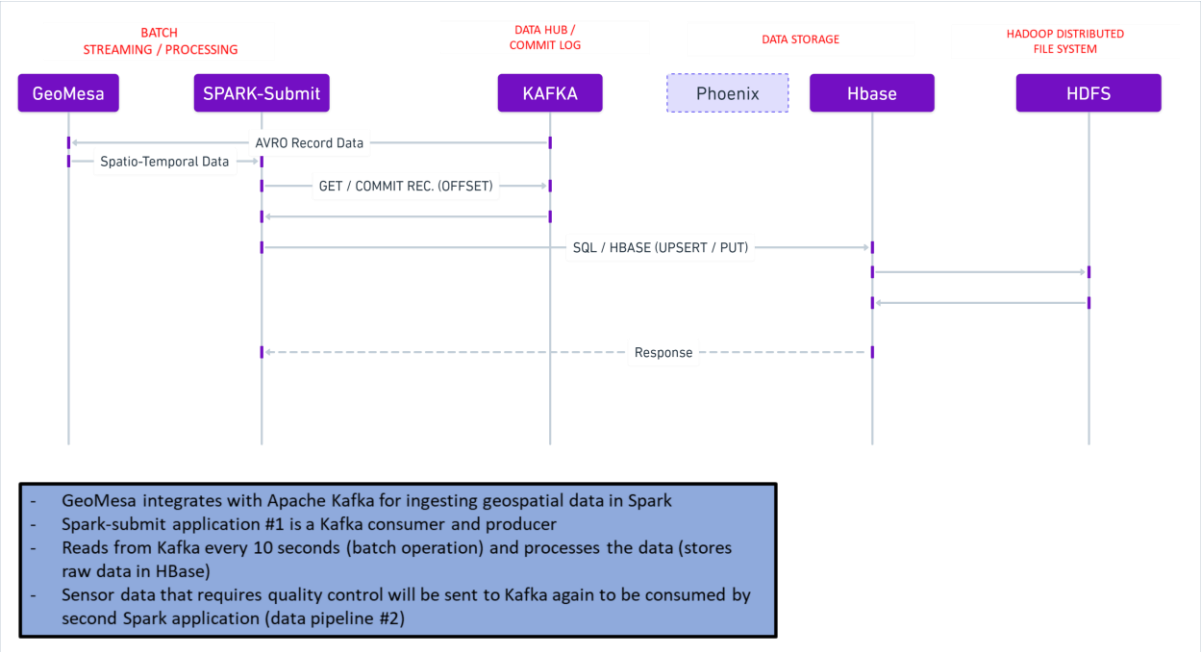


Figure 10. The sequence diagram for data pipeline #1 for incoming data storage.

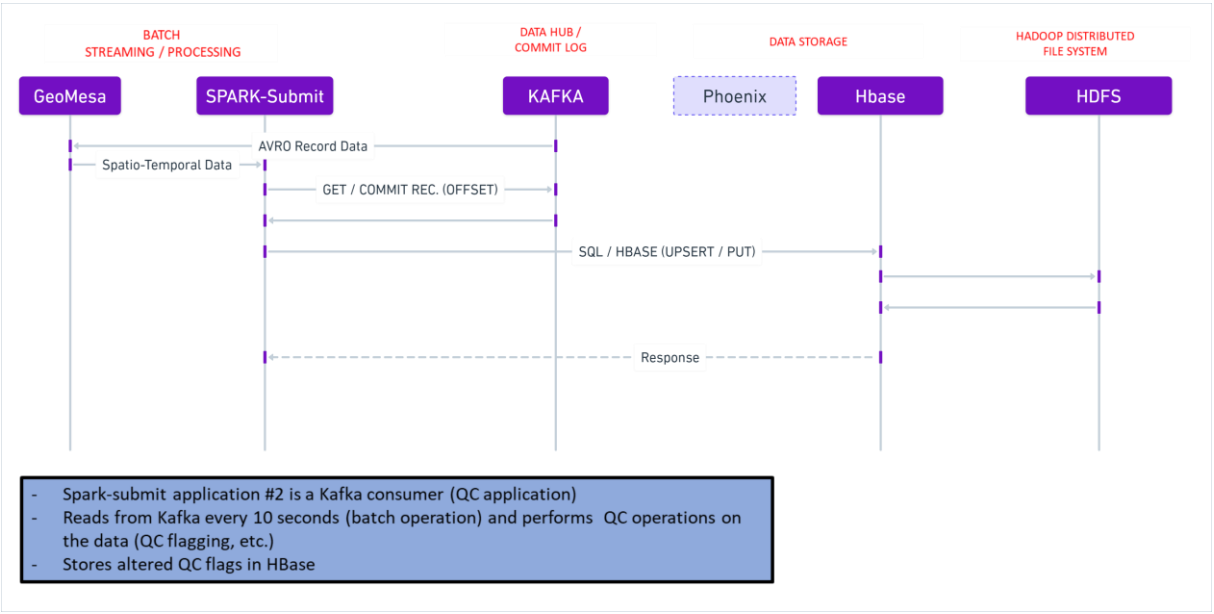


Figure 11. The sequence diagram for data pipeline #2 for quality control.

9.5.4 Persistence of RASTER Spatio-temporal RASTER Data and Quality Control

The results of the air quality and noise models are provided as GeoTiff files according to the OCG GeoTIFF standard (https://www.ogc.org/standard/geotiff/). Net4Cities Data Hub provides a storage and access solution based on PostGIS and GeoServer using the ImageMosaic plugin. In this context, a mosaic is a collection of raster images combined into a single dataset. This is useful for managing and visualizing large datasets covering wide geographic areas or multi-dimensional tiles.

The following sequence diagram (Figure 12) shows how the main process of saving a raster file to the Net4Cities datacenter works. Although authentication and authorization are not included in the diagram, they are part of the implemented system (token-based). In addition to load balancing, the interaction takes place between three components: the REST API, PostGIS (a plugin in PostgreSQL for working with spatial data), and the GeoServers REST API. The

modeling system sends its GeoTIFF files to the REST API, which PostGIS uses as a DBMS to index the ImageMosaic. In addition, a ZIP file containing the GeoTIFF files and datastore properties is sent to the GeoServer via the REST API so that the new image layer is available. For styling purposes (e.g., colors, labels), the system assigns an SLD and an XML-based style layer descriptor to the image layer, making it ready for visualization.

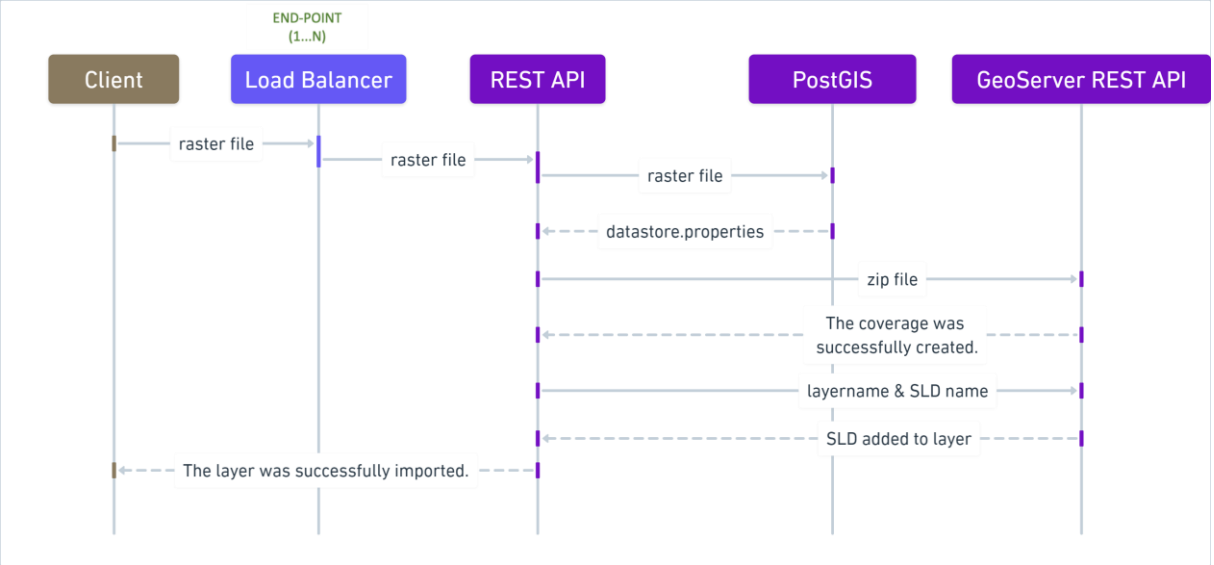


Figure 12. The sequence diagram for the data for incoming raster data storage.

In all these situations the system leverages technologies and frameworks to ensure performance, scalability and data integrity. Avros efficient serialization and schema development features play a role, in maintaining data compatibility. The distributed streaming platform offered by Apache Kafka delivers real-time data processing capabilities while guaranteeing latency and high throughput. Apache Sparks robust processing engine facilitates large-scale data transformations and analyses. GeoMesa seamlessly integrates functionalities with these components enhancing the system's capability to process and query geospatial datasets. PostgreSQL and HBase serve as scalable storage solutions each tailored for data types and access patterns. Lastly, GeoServer, along with platforms like Zeppelin and Jupyter, provides user interfaces for accessing, visualizing data and supporting a wide array of analytical tasks.

10. Deployment

Software systems rely on underlying hardware infrastructure to run effectively. Understanding this infrastructure is critical because it directly affects system performance and scalability. The deployed infrastructure supports:

- Source code repository and CI/CD pipelines will be done using GitLab. The deployment will be into multiple environments: development, testing, staging, and production.
- Utilization of Proxmox cluster for VM deployment, facilitating flexible resource allocation.
- Use of GlusterFS for storage in select environments, enhancing data reliability and access speed.
- VLAN-based network architecture ensuring direct communication between components.
- Planned scalability through additional Proxmox hosts, supporting both vertical and horizontal scaling strategies.

10.1 Identify Infrastructure Elements

Geographical Locations: All components are centralized at NILU's local data center in Kjeller, Norway.

Environments:

- Technical test: for testing Ansible scripts, etc.
- Dev: for testing newer versions of components and code.
- Stage: for testing what was in dev, but with the same dataset as for prod.
- Prod: for running the solution with the production dataset.

Computers/VMs/Containers:

- All components (such as HADOOP/YARN, HBASE, Confluent) will run in VMs on our Proxmox cluster.
- ZooKeeper also runs on servers with HADOOP/YARN.

Network Topologies: Components are on a single VLAN, communicating directly.

Other Infrastructure Elements:

- For some environments (tt, dev), storage on separate GlusterFS instance (SSD).
- Stage may also use GlusterFS (SSD).
- Production uses SSDs directly attached to Proxmox hosts, with VMs having dedicated storage on specific hosts.

10.2 Identify Software Elements

Identify Software Components:

- GlusterFS, HADOOP/YARN, HBASE, Confluent (Schema Registry, Kafka, Kafka REST).
- Front-end components like APIs managed by NILU DIGITAL.

Specify Deployment Units: Most components are downloaded as tar.gz files for Linux and run directly from extracted folders.

Deployment Environment Mapping: Components are deployed across all environments.

Deployment Configuration: Configuration details are handled by NILU DIGITAL.

10.3 Document the Mapping

Deployment Scripts and Automation:

- Ansible used to deploy HADOOP/YARN, HBASE, Confluent components.
- Ansible Semaphore manages Ansible scripts.

Consider Non-Functional Requirements:

- Performance: Testing is required once the environment is operational with production data.
- Scalability: Vertical scaling possible; horizontal scaling limited due to shared Proxmox cluster, though more hosts are being added.
- Security: Components communicate with SSL; Kerberos used for internal component communication (e.g., HADOOP/YARN).
- Maintainability: Updates challenging due to version compatibility; monitoring by Icinga/Zabbix combination.

11. Cross-cutting Concepts

This section outlines the principles, rules and potential solutions that are relevant to various aspects of the system. These overarching concepts help maintain consistency, uniformity and alignment with the vision for the system. They ensure that the system is developed and executed in a cohesive manner. Adhering to these guiding principles results in a level of unity within the system enhancing its reliability, ease of maintenance and expandability.

In the described data system architecture, several overarching concepts are discussed that are crucial across levels enhancing capabilities and ensuring smooth data operation and management. These key concepts include (illustrated in Figure 13):

- Data Serialization and Schema Management
- Real-Time Data Processing and Streaming
- Distributed Computing and Scalability
- Geospatial and Temporal Data Management
- Interactive Data Exploration and Visualization

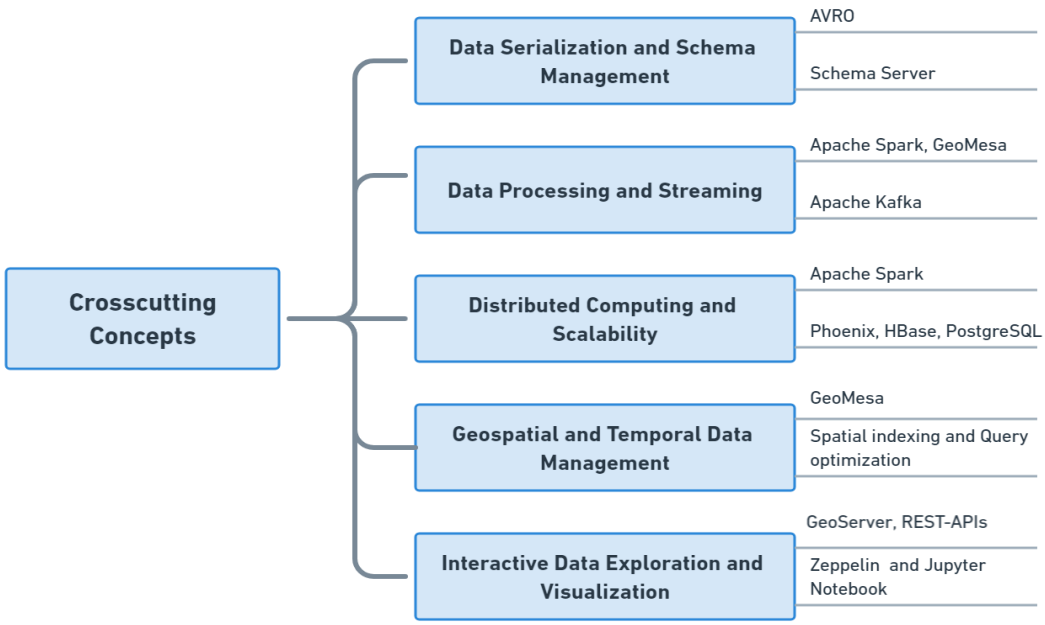


Figure 13. The crosscutting concepts.

11.1 Data Serialization and Schema Management

Using Avro for data serialization and managing schemas through a schema server are architectural concepts. Avro helps optimize data exchange by converting data into a form, which enhances storage and transmission efficiency while ensuring compatibility with evolving schemas. This is crucial in scenarios where data structures may change over time without disrupting existing data pipelines. The schema server plays a role in maintaining consistency by handling and versioning schemas for various data formats thereby enhancing data integrity and interoperability across different sources and storage systems.

11.2 Real-Time Data Processing and Streaming

In the realm of real-time data processing and streaming, Apache Kafka acts as the foundation for ingesting and streaming data within the architecture. It facilitates throughput low latency data transfer among distributed systems. It enables continuous flow of information from diverse sources to the processing layer. This setup ensures that the system can effectively handle streaming data, offer time capabilities and make timely decisions based on up-to-date information. Integrating Kafka with tools, like Apache Spark and GeoMesa enhances the architecture's capacity to process and analyze data streams efficiently while adapting to changing volumes and speeds of information.

11.3 Distributed Computing and Scalability

When it comes to distributed computing and scalability Apache Spark serves as a core element that exemplifies these concepts effectively.

Spark has the capability to efficiently manage datasets and handle computations by distributing tasks across multiple nodes. Its versatility in working with both batch and real-time processing modes offers flexibility for handling data types and workloads. This approach boosts performance. It also enables scalability by expanding horizontally as data volume increases. When coupled with storage solutions like HBase and PostgreSQL, Spark streamlines data retrieval and processing, ensuring a framework for the entire system.

11.4 Geospatial and Temporal Data Management

In managing temporal data, GeoMesa provides features tailored for spatial-temporal data management, which is particularly crucial in applications necessitating geospatial analysis and visualization. By integrating with Apache Kafka and other data processing frameworks, GeoMesa facilitates the storage, retrieval and querying of geospatial datasets. This functionality caters to applications such as monitoring, logistics and urban planning that rely on understanding spatial relationships and temporal patterns. With its indexing capabilities and query optimization features, GeoMesa strengthens the system's capacity to handle intricate geospatial queries while enabling profound data-driven insights for decision-making support.

11.5 Interactive Data Exploration and Visualization

For exploration of data visualization tools like Zeppelin Notebook and Jupyter Notebook add a layer of interactivity to the system by enabling analysis of datasets. These online notebooks create a space for data scientists and analysts to investigate data, execute algorithms and visualize outcomes in real-time. This idea promotes a process of data exploration and hypothesis testing, resulting in understanding and better communication of findings within the organization. By integrating GeoServer and REST API, these notebooks enable users to interact directly with processed data supporting agile decision-making processes that are well-informed.

These overarching concepts form the foundation of the system architecture being discussed, enhancing its abilities and ensuring its efficiency in handling types of data. They facilitate real-time processing, geospatial analysis, interactive data exploration and visualization in a web-based solution. By leveraging these concepts organizations can construct data infrastructures that not only handle data effectively but also uncover actionable insights. This drives innovation and strategic decision-making across domains.

12. Quality Requirements

Quality requirements define the non-functional characteristics that a system must have to meet user expectations and operational requirements. In the described data system architecture, several important quality requirements ensure its effectiveness, reliability, performance, and maintainability.

12.1 Quality Tree

A quality tree represents the hierarchy of functional requirements or quality attributes in a system outlining how these attributes are organized and prioritized to achieve overall system objectives.

12.1.1 Performance

The system needs to handle data volumes with minimal latency for real-time data streams. It should be capable of scaling to support increasing data loads and user needs without sacrificing performance. Moreover, interactive elements like REST APIs and data visualization tools should deliver responses promptly to ensure user satisfaction.

12.1.2 Reliability

Components such as Apache Kafka, HBase and PostgreSQL must exhibit resilience to guarantee data availability and consistency. Avro's serialization and schema management are essential for maintaining data coherence and ensuring compatibility across system components and versions.

12.1.3 Security

Ensuring the transmission and storage of information is crucial for compliance with regulations like GDPR. Implementing role-based access control for data APIs and interactive notebooks is necessary to prevent access and safeguard data confidentiality.

12.1.4 Maintainability

The architecture should be modularly designed with documentation for maintenance, updates and expansions. Keeping track of the performance of system components, like Kafka and Spark, through logging and monitoring is crucial for monitoring performance metrics identifying irregularities and aiding in problem solving.

12.1.5 Usability

When it comes to usability, having user-friendly interfaces such as Zeppelin and Jupyter notebooks is key for data exploration and visualization. Providing documentation and tutorials to both users and administrators is essential for system integration and operation.

12.2 Quality Scenarios

12.2.1 Scenario 1: High Throughput Data Ingestion

The system needs to handle 10,000 data points per second from instrument APIs with minimum latency (less than 100 milliseconds) when ingesting data into Kafka. Monitoring the average time taken to ingest and process instrument data will ensure it meets the latency standards. Load testing tools will be used to simulate peak data loads to test if Kafka and Spark can maintain the throughput without performance issues.

12.2.2 Scenario 2: Real-Time Query Response

Real-time query responses for GeoMesa spatial data queries should deliver results within one second for queries. Performance testing, under load conditions, will be conducted to validate if GeoMesa queries meet the latency requirements. Monitoring tools integrated with GeoMesa will be used to verify response times during testing.

12.2.3 Scenario 3: Scalability

The system needs to be able to grow to handle a 50% increase in data volume within six months without needing changes in its design. We will check this by monitoring how the system uses its resources (like CPU, memory, disk I/O) during times and predicting how well it can grow based on trends. To confirm this, we will do stress tests to see if tools like Kafka, Spark and storage systems can handle data without losing performance.

12.2.4 Scenario 4: System Monitoring and Maintenance

Centralized logging and monitoring must be implemented for Kafka, Spark, and database systems to proactively detect and respond to system failures. This requirement will be measured by setting up monitoring dashboards to track system performance metrics such as throughput, latency, and error rates in real-time. Verification will involve conducting incident response drills to test the effectiveness of monitoring alerts and procedures for handling system failures and performance bottlenecks.

13. Risks and Technical Debts

Every intricate data system structure comes with risks that could affect its operation, efficiency and overall dependability. By pinpointing these, risks strategies can be put in place to lessen disruptions.

In the distributed computing environment, utilizing tools such as Apache Kafka and HBase introduces significant challenges related to **data consistency and integrity**. Due to the distributed nature of such systems, there is an enhanced risk of encountering issues with data consistency, particularly when managing large volumes of data or during system failures. Inaccurate data can impact negatively on business functions and diminish user confidence in the system. To address such risks effectively, it is crucial to establish robust data validation mechanisms between Kafka producers and consumers, utilize error-handling capabilities strategically, and routinely verify data integrity through automated checks alongside manual validation procedures.

Considering the **scalability and performance**, tools, such as Apache Spark and Kafka are inherently designed to handle scalability efficiently. However, sudden increase in data volumes or unforeseen spikes in user activity can still strain system resources. This can lead to a decline in performance, resulting in longer processing times for data, slower responses to queries, and potential system failures during peak usage periods. To mitigate these risks effectively, it is essential to conduct thorough load testing to simulate various scalability scenarios, optimize cluster configurations for resource efficiency, and implement auto-scaling features to dynamically adjust resources based on real-time workload demands.

Another major concern is **security vulnerabilities** due to the system's data inputs and outputs. These aspects expose the system to security risks such as access, data breaches and denial of service attacks. Breaches could lead to data leaks, loss of information, noncompliance with regulations and damage to reputation. Strategies for mitigating these risks involve implementing authentication and authorization mechanisms for API endpoints and data storage systems,

regularly updating software components, conducting security audits and penetration tests as well as ensuring encryption for data in transit and at rest.

13.1 Technical Debts in the System Description

The section on Technical Debts in the System Description discusses how technical debt accumulates from incomplete design choices made during the development and implementation phases. These decisions can result in lower system performance, maintenance challenges and delays in feature enhancements.

Architectural complexity serves as a type of debt that emerges from integrating various technologies, like Apache Kafka, Spark, GeoMesa and different storage solutions. The intricacy of these situations can result in increased debt because of the connections between components causing challenges in diagnosing and fixing issues and potentially lengthening development phases and deployment schedules. To manage this complexity effectively, it is essential to review the architecture, refactor code and configurations to simplify interactions and update documentation for clarity during future maintenance and enhancements.

Issues with **maintaining code and configurations** also contribute to debt. Quick development cycles and evolving requirements often lead to compromises on code quality and configuration management practices resulting in an accumulation of debt over time. Source-codes, outdated configurations and dependencies on libraries can compromise the stability, scalability and maintainability of your system. To address this issue, it is crucial to embrace integration and delivery (CI/CD) practices for automating testing and deployment processes conduct code reviews prioritize refactoring initiatives and keep documentation up to date.

Furthermore, **data governance** issues pose risks related to debt. Issues like inconsistent schema management or insufficient data validation procedures can lead to debt associated with data integrity and compliance. Noncompliance with regulations (such as GDPR) or internal data standards may result in liabilities, data breaches or disruptions in operations. To tackle this issue, it is crucial to set up defined data management rules and protocols, automate data validation tasks when feasible carry out checks to uphold standards, and offer continuous training to individuals handling data management and oversight.

14. Glossary

Table 6. Table of key terms used in this template.

Term	Description
arc42	Template we used for this document, which are used describe system architecture related to software (https://arc42.org/overview).
Avro	A data serialization system that is used for efficiently storing and transferring data, especially used within big data (https://avro.apache.org/).
C#	Object-oriented programming language developed by Microsoft and is part of the .NET framework (https://dotnet.microsoft.com/en-us/languages/csharp).
GeoMesa	An open-source large-scale geospatial software package (https://www.geomesa.org/)
GeoServer	An open-source server for sharing geospatial data (https://geoserver.org/)
GeoTIFF	A file format for storing georeferenced raster imagery.
GlusterFS	A scalable network filesystem.
HADOOP	A framework for distributed storage and processing of large data sets (https://hadoop.apache.org/)
Hbase	An open-source, distributed database (https://hbase.apache.org/)
HTTP GET	An HTTP request method is used to retrieve data from a server.
HTTP POST	An HTTP request method is used to send data to a server.
Icinga	An open-source monitoring system (https://icinga.com/)
ImageMosaic	A tool for creating mosaics of geospatial imagery.
Java	A high-level programming language used for building applications https://www.java.com/en/)
Jupyter Notebook	An open-source web application for creating and sharing documents with runtime code and visualizations (https://jupyter.org)
Kafka	A distributed event streaming platform (https://kafka.apache.org/)
Kerberos	A network authentication protocol (https://web.mit.edu/kerberos/)
Phoenix	An open-source SQL query engine for Hbase (https://phoenix.apache.org/)
PostGIS	A spatial database plugin for PostgreSQL (https://postgis.net/)
PostgreSQL	An open-source relational database management system (https://www.postgresql.org/)
Proxmox	A server virtualization management platform (https://www.proxmox.com/)
Scala	A programming language (https://www.scala-lang.org/)
Schema Server	A server that manages schemas for Avro data.
Spark	A big data processing engine
YARN	A resource management layer for Hadoop (https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html)
Zabbix	An open-source monitoring software (https://www.zabbix.com/)
Zeppelin Notebook	An open-source web application for creating and sharing documents with runtime code and visualizations (https://zeppelin.apache.org/)
ZooKeeper	A centralized service for maintaining configuration information for distributed synchronization (https://zookeeper.apache.org/)

Conclusions

The design of the Net4Cities Data Hub effectively achieves its objectives by creating a scalable and flexible system capable of retrieving, processing, storing and making the relevant data from the Net4Cities project accessible to a wide range of stakeholders. The layered approach to data ingestion, storage, processing and delivery ensures that data integrity and accessibility are maintained consistently. This architecture not only meets the requirements of the Net4Cities project but also establishes a solid groundwork for future improvements and integrations. The results of this output exhibit an organized strategy for managing and analyzing data, which is essential for understanding urban environmental conditions and providing valuable insights for stakeholders. As the project advances, the documented architectural choices will support development and upkeep, ensuring that the system stays in line with evolving needs. Subsequent tasks will expand on this framework by enhancing data processing capabilities, improving interfaces and incorporating data sources to further advance the objectives of the Net4Cities project in fostering more sustainable urban environments.

Acknowledgment

We acknowledge the use of ChatGPT-4o (https://chat.openai.com/) in the following way:

(i) to find relevant academic literature to help us to understand the arguments
(ii) to refine the academic language.

Acronyms

Table 7. Table of acronyms used in this template.

Acronym	Meaning
ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
BC	Black Carbon, which is a component of particulate matter (PM).
CH4	Methane
CI/CD	Continuous Integration/Continuous Deployment, a practice for automating software development and deployment.
CO2	Carbon Dioxide
GDPR	General Data Protection Regulation
GHG	Greenhouse Gases
GIS	Geographic Information Systems
HDFS	Hadoop Distributed File System
HTTP	Hypertext Transfer Protocol, the foundation of data communication on the web.
JSON	JavaScript Object Notation, a lightweight data interchange format.
LDSA	Lung-depositing surface area of particles.
LULC	Land Use and Land Cover
LULC	Land Use/Land Cover, a classification of land based on its use and vegetation.
N2O	Nitrous Oxide

N4C	Net4Cities
NH3	Ammonia
PM10	Particulate Matter 10 micrometers or less in diameter.
PNC	Particle Number Concentration, a measure of the number of particles in a given volume of air.
QC	Quality Control
QM	Quality Management
RDD	Resilient Distributed Dataset, a fundamental data structure of Apache Spark.
REST API	Representational State Transfer API and is a set of guidelines for building scalable web services.
SLD	Styled Layer Descriptor, an XML schema for styling of maps
SLM	Sound Level Meter
SLM	Service Level Management, a practice for managing service levels in IT.
SQL	Structured Query Language, a standard language for managing relational databases.
SSD	Solid State Drive, a storage device.
SSL	Secure Sockets Layer, a protocol for securing communications on the internet.
TLS	Transport Layer Security, a protocol for securing communications on the internet.
VLAN	Virtual Local Area Network
VM	Virtual Machine
VOC	Volatile Organic Compounds
WFS	Web Feature Service
WMS	Web Map Service
XML	Extensible Markup Language is a markup language and file format for storage and transferring data.

References

1. Starke, G., Simons, M., Zörner, S., & Müller, R. D. (2019). arc42 by Example: Software architecture documentation in practice. Packt Publishing Ltd.
2. European Commission, Directorate-General for Environment, Nagl, C., Bleeker, A., Ntziachristos, L. et al., Systematic assessment of monitoring of other air pollutants not covered under Directives 2004/107/EC and 2008/50/EC – With a focus on ultrafine particles, black carbon/ elemental carbon, ammonia and methane in ambient air, Publications Office of the European Union, 2022, **https://data.europa.eu/doi/10.2779/691266**
3. Garrido Salcedo, J. C., Mosquera Lareo, B. M., Echarte Puy, J., & Sanz Pozo, R. (2019, September). Management Noise Network of Madrid City Council. In INTER-NOISE and NOISE-CON Congress and Conference Proceedings (Vol. 259, No. 8, pp. 1700-1711). Institute of Noise Control Engineering.
4. Giovannini, L., Ferrero, E., Karl, T., Rotach, M. W., Staquet, C., Trini Castelli, S., & Zardi, D. (2020). Atmospheric pollutant dispersion over complex terrain: Challenges and needs for improving air quality measurements and modeling. Atmosphere, 11(6), 646.
5. Guo, D.; Onstein, E. State-of-the-Art Geospatial Information Processing in NoSQL Databases. ISPRS Int. J. Geo-Inf. 2020, 9, 331. https://doi.org/10.3390/ijgi9050331
6. Raptis, T.P.; Cicconetti, C.; Falelakis, M.; Kalogiannis, G.; Kanellos, T.; Lobo, T.P. Engineering Resource-Efficient Data Management for Smart Cities with Apache Kafka. *Future Internet* 2023, *15*, 43. https://doi.org/10.3390/fi15020043
7. Humphreys, E. (2008). Information security management standards: Compliance, governance and risk management. information security technical report, 13(4), 247-255.
8. Vohra, D., & Vohra, D. (2016). Apache avro. Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools, 303-323.
9. Obe, R. O., & Hsu, L. S. (2017). PostgreSQL: up and running: a practical guide to the advanced open source database. " O'Reilly Media, Inc.".
10. Vora, M. N. (2011, December). Hadoop-HBase for large-scale data. In Proceedings of 2011 International Conference on Computer Science and Network Technology (Vol. 1, pp. 601-605). IEEE.
11. Albrecht, J. (2017). Using Web-Based Notebooks For Blended-Learning In Computer Science. In EDULEARN17 Proceedings (pp. 5714-5720). IATED.
12. Srivastava, A., Bhardwaj, S., & Saraswat, S. (2017, May). SCRUM model for agile methodology. In 2017 International Conference on Computing, Communication and Automation (ICCCA) (pp. 864-869). IEEE.