Scientific Research Publishing

Journal of Data Analysis and Information Processing, 2025, 13(4), 467-486
https://www.scirp.org/journal/jdaip
ISSN Online: 2327-7203
ISSN Print: 2327-7211

Data Lakes as a Centralized Integration Layer in Enterprise Environments: Approaches and Benefits for Scalability and Performance

Carlos Diego Cavalcanti Pereira

CESAR School, Recife, Brazil
Email: cdcp@cesar.school

How to cite this paper: Cavalcanti Pereira, C.D. (2025) Data Lakes as a Centralized Integration Layer in Enterprise Environments: Approaches and Benefits for Scalability and Performance. *Journal of Data Analysis and Information Processing*, **13**, 467-486.
https://doi.org/10.4236/jdaip.2025.134027

Received: July 11, 2025
Accepted: September 26, 2025
Published: September 29, 2025

Abstract

Enterprise application integration encounters substantial hurdles, particularly in intricate contexts that require elevated scalability and speed. Transactional applications directly accessed by many systems frequently overload databases, undermining process efficiency. This paper examines the utilization of data lakes—historically used for data analysis—as a centralized integration layer that accommodates various temporalities and consumption modalities. The suggested method diminishes system interdependence and the burden on transactional databases, enhancing scalability and data governance in both monolithic and distributed frameworks.

Keywords

Application Integration, Data Lakes, Data Governance

1. Introduction

In contemporary corporate environments, data integration across applications is crucial for effective operations and informed decision-making. The growth of IT architectures and the shift toward more distributed systems present new issues in sustaining efficient, reliable, and scalable data flows between systems [1]. These problems are especially significant in complex architectures that integrate legacy monolithic systems with contemporary microservices. In all instances, direct and recurrent access to centralized transactional databases may constitute a significant bottleneck, impacting system performance, availability, and scalability.

To mitigate these challenges, many companies allocate resources towards establishing resilient data infrastructures, such as data lakes. Initially utilized as centralized repositories for substantial amounts of un-structured and semi-structured

data, data lakes have demonstrated further utility: the capacity to function as a centralized integration layer for diverse data consumption types, thereby diminishing systems' reliance on transactional databases [2].

The significance of a centralized data integration layer is particularly evident in enterprise contexts that combine diverse data sources and timeframes, encompassing historical records and real-time data [3]. Instead of linking each application directly to another application's database, the data lake serves as a middleman, offering a centralized interface that mitigates the complexity of data access and transformation. The data lake, enhanced by optimal storage and streaming technologies, may fulfill several requirements, including historical data for analytics, real-time data for operations, and asynchronous data for background processing [4].

This article presents a data lake architecture that transcends its conventional function as an analytical repository, assuming the role of an integration layer for both distributed and monolithic enterprise applications. Investigating the data lake in this manner can assist enterprises in scaling operations, alleviating strain on transactional systems, and allowing applications to utilize data with varying temporalities—without necessitating extensive restructuring.

The subsequent sections of this work are structured as follows: Section 2 addresses the primary obstacles in the integration of data within enterprise applications. Section 3 evaluates current architectural methodologies. Section 4 delineates the proposed Data Lake design. Section 5 delineates its function as a centralized integration layer for both distributed and monolithic applications, encompassing its benefits and limitations. Section 6 finishes with conclusive reflections.

2. Challenges in Enterprise Application Data Integration

Data integration across enterprise applications represents a primary challenge for organizations, particularly in contexts where the complexity of IT infrastructures escalates due to a combination of monolithic and distributed systems [1]. In structures that integrate historical systems with modern microservices-based applications, it is common for applications to require access to data from other systems, resulting in dependencies that affect performance, scalability, and data governance.

2.1. Overload on Transactional Databases

In several organizations, transactional databases, including those that underpin ERP, CRM, or other business systems, are central to operations and must accommodate both operational requirements and application integration. This heavy utilization results in various performance and scalability challenges, potentially undermining system stability. The simultaneous access of numerous applications to a transactional database for reading or modifying data substantially elevates the database load. This overload arises from a confluence of complex queries, frequent updates, and rapid read operations, coupled with competition for processor and

memory resources. Transactional databases were not engineered to manage the concurrent traffic produced by many applications [5]. The result of such heavy usage is diminished performance, prolonged response times, and an increased likelihood of transaction failures, adversely affecting user experience and operational continuity.

A key component in transactional database overload is that numerous systems were engineered solely to facilitate the everyday functions of a particular application, rather than to function as centralized repositories for shared data. In high-consumption scenarios—where applications execute frequent and complicated queries—the database becomes excessively burdened, adversely affecting both external requests and the internal operations of the core application. Excessive utilization of resources, including CPU processing, memory, and disk I/O, constrains system responsiveness and may result in significant bottlenecks, particularly during peak periods [6].

Furthermore, excessive load on transactional databases imposes scalability constraints on corporate systems, as illustrated in **Figure 1**. A central database serving as the sole access point for numerous applications creates a bottleneck, hindering the efficient growth of the IT infrastructure. This singular dependency restricts the enhancement of processing and storage capabilities, as transactional databases frequently encounter scalability constraints—especially when implemented in monolithic systems. These limits diminish the organization's adaptability to emerging demands and escalate infrastructure and maintenance expenses.

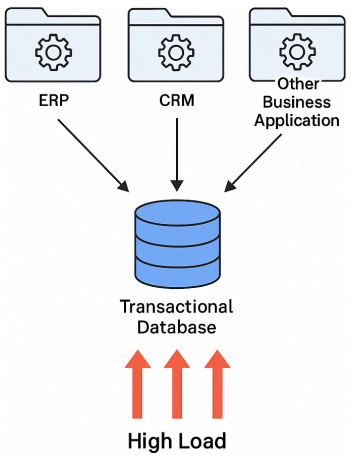

Figure 1. Overload caused by multiple applications consuming transactional systems. Source: Author (2025).

To reduce overload issues, many companies implement workaround strategies, such as database replication to divide the load and the utilization of intermediary caches. Nonetheless, these methodologies sometimes provide new issues, including the preservation of consistency across replicated data and the management of the complexities associated with many data copies. Caching methods do not fully resolve the issue, as they do not eradicate the direct reliance of applications on

transactional databases; they merely alleviate the impact to some extent.

Modern approaches increasingly employ Change Data Capture (CDC) procedures [3] to capture and propagate database modifications in near real-time. This technique seamlessly interfaces with the ingestion layer outlined in Section 4.1, facilitating continuous synchronization between transactional systems and the Data Lake without burdening operational databases.

2.2. Dependency and Coupling between Services

The direct dependency between services and transactional databases results in tight coupling, limiting the expansion and scalability of corporate systems. This coupling occurs when various applications, whether monolithic or distributed as microservices, directly access the same database to exchange information or execute integrations. In complex enterprise environments, this strategy generates interdependencies that complicate system management, maintenance, and evolution, ultimately constraining the organization's capacity to swiftly adapt to evolving business requirements [7].

In microservices architectures, each service is engineered to operate independently, possessing its own business logic and ideally its own database. This autonomy enables the creation, implementation, and scalability of each service independently. However, when various services require direct access to the same data, that autonomy is undermined. Rather than functioning independently, these services rely on each other for vital data access, resulting in significant coupling that complicates modifications to one service without affecting the others [5].

This coupling presents operational issues, as changes to the shared database structure—such as changes to tables or indexes—can directly affect all dependent services. In a system whose services are linked to a singular database, modifying one component often requires a comprehensive evaluation of the repercussions on all other associated services. This coordination procedure not only elevates the time and cost associated with modifications but also amplifies the potential of errors and inconsistencies throughout implementation, hence diminishing organizational agility.

Coupling impacts architectural scalability by constraining the capacity to scale each service autonomously. In a situation where numerous services utilize the same database, scaling an individual service does not entirely resolve the performance problem. The database serves as a singular access point and may create a bottleneck, particularly during high-load scenarios. The reliance on a centralized database obstructs the architecture's capacity to flexibly adjust to emerging demands and impedes its efficiency in responding to traffic surges [8].

In addition to scaling constraints, service coupling further complicates the execution of resilience and disaster recovery techniques, as seen in **Figure 2**. In tightly connected designs, a failure in one service that directly accesses the database might adversely affect other dependent services, resulting in a cascading effect that jeopardizes the availability of the entire system. The issue is intensified

when numerous services utilize the same database connection and vie for identical resources, including processing power and memory. In distributed contexts, where availability and resilience are essential, coupling is a factor that considerably diminishes system reliability [9].

To reduce service coupling as shown in Figure 2 and enhance autonomy and scalability, several architectures implement strategies like data replication or the utilization of integration APIs. Nonetheless, although operational, these solutions present further complications. Data replication necessitates continuous synchronization to sustain consistency among numerous copies, hence augmenting infrastructure complexity and expense. Integration APIs facilitate communication between services; however, they do not entirely eradicate reliance on the database, particularly when the data is essential and requires real-time updates.

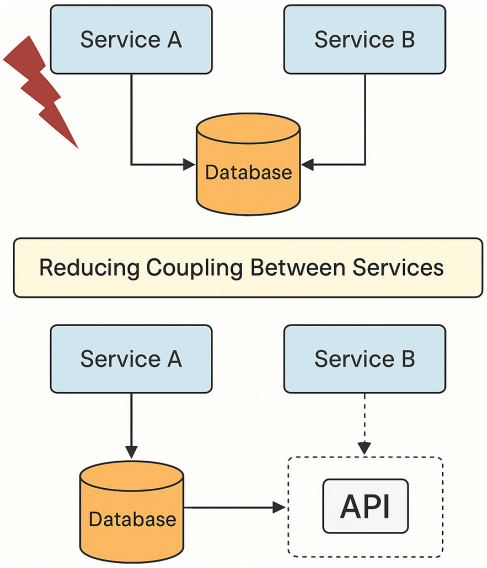

Figure 2. Services consuming and competing with transactional applications. Source: Author (2025).

2.3. Data Consistency and Integrity

Data consistency and integrity are essential components for the reliability of enterprise systems, especially in contexts where several applications access and alter crucial data. The interaction of multiple applications with the same transactional database considerably increases the risk of inconsistencies, jeopardizing data integrity and complicating governance and auditing procedures. In intricate enterprise situations, preserving data integrity is crucial to guarantee that information is reliable and that strategic decisions are founded on precise data [9].

This issue is particularly critical in distributed contexts, where many applications or microservices can independently access and change data. When applications utilize a centralized database, there exists an inherent risk that simultaneous activities may lead to discrepancies. A prevalent instance is the emergence of race circumstances, wherein multiple concurrent actions endeavor to alter the same

record, resulting in an inconsistent data state. This situation is intensified when activities lack synchronization or fail to adhere to atomic transaction standards, complicating the recovery and rectification of erroneous data [7].

The absence of data consistency directly impacts user experience and the efficacy of business processes. In an e-commerce system reliant on current inventory data, inconsistencies may lead to problems such as the sale of out-of-stock items or errors in order processing. This issue not only disrupts operations but also adversely affects the customer's view of the system's and company's reliability.

A significant difficulty concerning data consistency and integrity is the replication and divergence of information across services. In distributed systems, it is typical for several services to retain their own copies of identical data to enhance efficiency and minimize reaction times (**Figure 3**). This method results in data duplication and increases the risk of divergence, as a modification in one data instance may not be promptly mirrored in others, particularly when services function asynchronously. This may lead to divergent interpretations of the data, undermining information accuracy and obstructing decision-making.

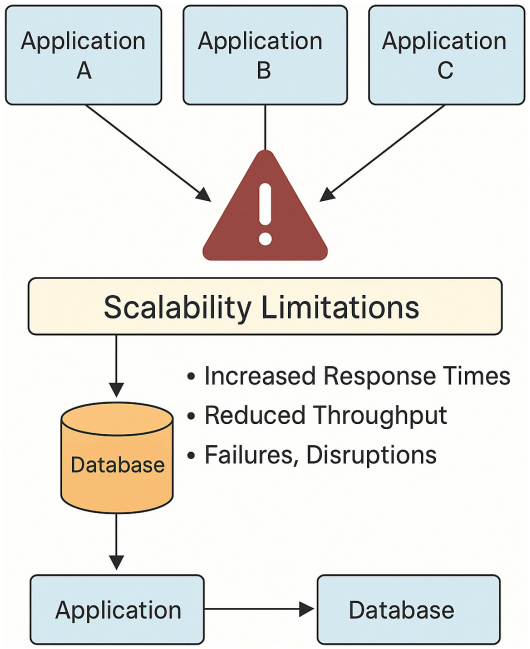

Figure 3. Scalability limitations in coupled architectures. Source: Author (2025).

2.4. Scalability and Performance

Scalability and high performance are critical in enterprise systems to effectively facilitate daily operations, particularly in businesses that are managing substantial data volumes and continuous traffic. In environments where numerous applications interact with a centralized transactional database, conventional integration architectures become impediments, as these systems were not engineered to accommodate escalating volumes of concurrent queries and extensive read/write transactions. This constraint restricts infrastructure scalability and impairs system perfor-

mance, affecting user experience and the efficacy of enterprise operations [8].

A primary scalability constraint in architectures utilizing shared transactional databases is that the centralized database serves as the sole access point for all applications and services. As workload escalates, this centralized point experiences resource contention, including CPU processing, memory, and I/O capability. In situations involving elevated amounts of concurrent transactions, the database rapidly attains its capacity threshold, resulting in prolonged response times, diminished operational throughput, and, in severe instances, system breakdowns or outages. This issue intensifies during peak usage periods, as heightened demand exacerbates the constraints of conventional architecture [7].

A further scalability barrier is the complexity of separately scaling the diverse components of the architecture. In a closely coupled design, where numerous applications directly contact a centralized database, expanding an individual service fails to address the issue, as the database persists as a singular point of dependency—and therefore, a bottleneck. Despite scaling the service to accommodate increased demand, it will remain limited by the capacity of the centralized database. This integration paradigm undermines horizontal scalability—the capacity to augment service instances to accommodate heightened demand—and hinders the organization's agility in responding to evolving business requirements and shifts in the operational landscape [5].

The performance of the system is affected by the latency linked to simultaneous requests and transactions. In a centralized database, numerous applications vie for resources to execute queries and updates, resulting in heightened latency and diminished operational performance. This latency is especially detrimental in microservices systems, where reaction time is essential for maintaining a seamless and efficient user experience. Elevated latency might result in subpar user experiences and performance complications that directly impact the applications' capacity to fulfill end-user requirements.

2.5. Data Governance and Security

Data governance and security are two essential components of data integration in business environments. In systems where multiple applications directly interface with transactional databases, managing and safeguarding data access becomes challenging, increasing the constraints of who can see, change, or distribute information. This problem intensifies when multiple teams, systems, and external partners access the same database, increasing the possibility of unauthorized access, inappropriate manipulation, and sensitive data leaks [3].

The lack of a centralized integration layer affects the implementation of data control and monitoring policies from a governance perspective. Each application interfacing with the transactional database may function at varying access levels, resulting in a lack of uniformity and insufficient monitoring. This fosters an environment in which data may be easily modified or inconsistently utilized across applications, compromising information accuracy and heightening the risk of

non-compliance with data protection regulations, such as the GDPR (General Data Protection Regulation—EU) and the LGPD (General Data Protection Law—Brazil). Effective data governance requires supervision of the source, accessibility, integrity, and utilization of information across its lifecycle—an endeavor more challenging in decentralized and closely integrated infrastructures [7].

Data security is compromised when several applications directly access transactional databases. This behavior increases the number of system entry points, broadens the attack surface, and complicates the execution of holistic security measures. In an architecture where multiple applications are directly linked to the database, centralized monitoring and management of access become more challenging, hence increasing the risk of unauthorized access and data misuse. This situation is particularly alarming in sectors like healthcare, banking, and the public domain, where data is extremely sensitive and requires stringent protection to avert leaks and hacks. A significant security consideration is audit control. In architectures where different applications directly access data, it becomes challenging to trace and audit who accessed, updated, or viewed particular information. This diminishes transparency regarding data utilization and compromises the organization's capacity to identify and address suspicious or inappropriate conduct. The absence of a centralized governance framework hampers the execution of data retention and disposal rules, which are crucial for ensuring that information is maintained only as long as necessary and safely deleted when no longer required [8].

3. Architectural Approaches for Enterprise Application Integration

Enterprise application integration is crucial for achieving interoperability and efficiency in corporate processes. Throughout the years, numerous architectural methodologies have emerged to enable system integration, spanning from monolithic solutions to contemporary networked designs. The literature and industry employ distinct integration methodologies in corporate environments, focusing on service-oriented architectures, microservices, and integration bus-based solutions. Each technique has distinct advantages and disadvantages, particularly with data integration and the accommodation of varying temporalities and consumption levels.

3.1. Service-Oriented Architecture (SOA)

In a Service-Oriented Architecture (SOA), system functionalities are organized as autonomous services that can be repurposed across various contexts. These services are made accessible via standardized interfaces, such as XML and SOAP (Simple Object Access Protocol), which enable communication between diverse systems and promote application integration inside large enterprises [10].

Although SOA offers flexibility by enabling various systems to utilize services without direct access to transactional databases, it also has constraints. Communication in SOA frequently utilizes substantial protocols like SOAP, which can

elevate latency and diminish performance. The adoption of SOA necessitates strong governance to prevent redundancy and guarantee uniform service utilization. The coupling of services in Service Oriented Architecture (SOA) is a significant issue, particularly in contexts with rapidly evolving business requirements, which can render the system more inflexible and challenging to scale [11].

3.2. Microservices Architecture

Microservices architecture represents a further development of SOA, aimed at resolving certain coupling and scalability challenges identified in conventional integration structures. In a microservices architecture, each service is accountable for a distinct business function and is created, deployed, and scaled autonomously. Every microservice possesses an individual database, facilitating total data segregation and enhancing flexibility and horizontal scalability [12].

Microservices empower development teams to operate independently, enhancing agility and streamlining the integration of new features. This autonomy presents difficulties in data integration. As each microservice possesses an independent database, it is imperative to synchronize information among them to maintain data consistency and the integrity of business processes. To resolve this, microservices architectures frequently employ events and messaging systems—such as Kafka or RabbitMQ—to facilitate asynchronous data transmission between services. This method may become intricate, especially when managing substantial data volumes or requiring real-time consistency [5].

3.3. Enterprise Service Bus

The Enterprise Service Bus (ESB) is an integration architecture intended to facilitate communication among systems inside enterprise contexts. The ESB functions as a centralized bus enabling many applications to connect and exchange messages through designated adapters. It enables communication among diverse systems and permits the centralization of business rules and data transformations, hence streamlining integration and minimizing the necessity for adaptation in each connected application [13].

The ESB provides substantial benefits by minimizing inter-system connection and centralizing integration logic. Nonetheless, it possesses constraints, particularly regarding scalability and performance. The ESB serves as a singular integration point, potentially becoming a bottleneck in situations characterized by elevated data volumes and frequent demand surges. Reliance on a singular centralized bus constrains architectural resilience, as a breakdown in the ESB could affect all interconnected systems. The centralization renders the ESB less appropriate for dispersed environments requiring high availability and low latency [9].

3.4. Asynchronous Messaging and Data Streaming

To address scalability and coupling issues inherent in conventional methods, numerous contemporary systems implement asynchronous messaging and data

streaming solutions. Technologies such as Apache Kafka and RabbitMQ facilitate asynchronous communication between services, permitting each service to process data at its own velocity and diminishing reliance on direct connections to transactional databases. In distributed, high-load systems, asynchronous messaging mitigates delay and enhances resilience, as a failure in one service does not impede the operation of others.

These technologies offer an efficient answer for situations where eventual consistency is permissible and services can function independently. Nonetheless, asynchronous messaging poses issues, especially in scenarios necessitating real-time consistency or when substantial data quantities must be exchanged between services. The administration of historical data and the integration of information across systems continue to pose challenges in exclusively event-driven contexts, since the absence of a centralized data perspective complicates governance [4].

4. Data Lake Architecture

Data Lake architecture Figure 4 has emerged as a proficient option for aggregating substantial quantities of data from diverse sources, keeping it in a unified repository that enhances accessibility and analysis. Data lakes were initially created to provide extensive data analytics and the assimilation of diverse information—structured, semi-structured, and unstructured—aiming to deliver a comprehensive and unified perspective of organizational data. The adaptability of a data lake enables it to accommodate a spectrum of data, as presented in 4, ranging from unprocessed to processed information, thus serving as an essential element in fulfilling the analytical and operational requirements of contemporary organizations [8].

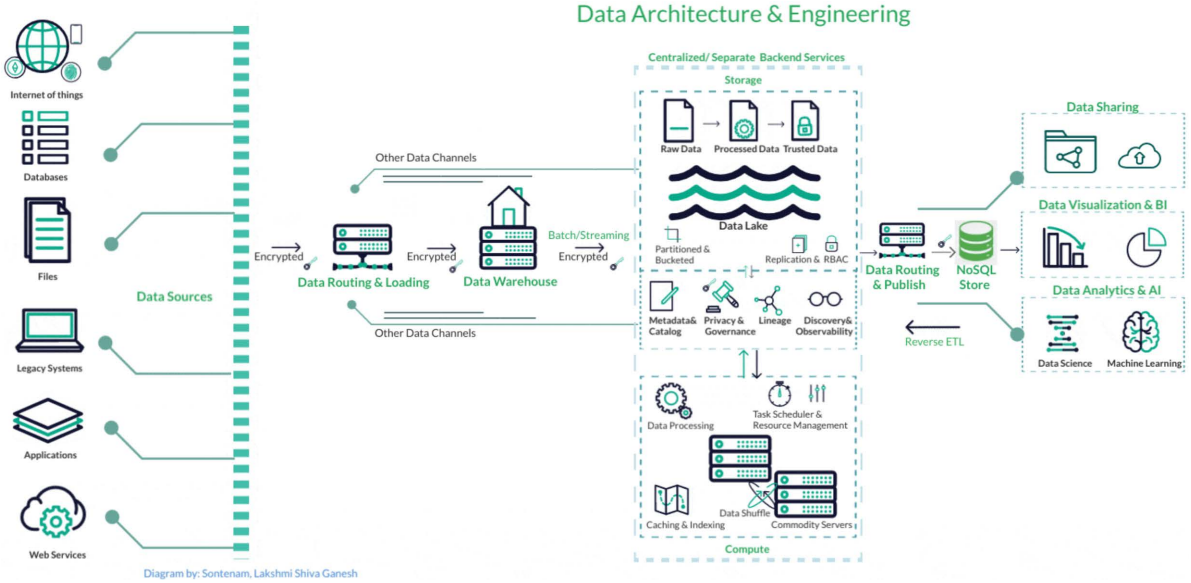

Figure 4. Reference architecture of a Data Lake (Data Architecture & Engineering, Lakshmi Shiva Ganesh Sontenam. Accessed on October 11, 2024. Available at: https://shivaga9esh.medium.com/data-architecture-engineering-22164c8dbd43).

4.1. Data Ingestion Layer

The ingestion layer is tasked with aggregating data from multiple sources and placing it into the data lake. Sources may encompass transactional systems, IoT devices, social media platforms, log data, and various enterprise applications. This layer must be engineered to accommodate a diverse array of data formats, encompassing structured files (e.g., tables and CSVs), semi-structured formats (e.g., JSON and XML), and unstructured data (e.g., photos, videos, and text files).

The ingestion layer may employ technologies such as ETL (Extract, Transform, Load) and ELT (Extract, Load, Transform) to facilitate continuous data intake with varying temporalities, processing data either before or subsequent to its entry into the Data Lake. Streaming solutions like Apache Kafka and AWS Kinesis facilitate low-latency ingestion for realtime data, guaranteeing the Data Lake receives near real-time changes [4].

4.2. Data Storage Layer

The storage layer is the repository for raw, refined, and converted data within the Data Lake. This layer must be scalable and economical, as the data lake accommodates vast quantities of data from various sources and across diverse timeframes. Data Lake storage is generally constructed on distributed file systems like Amazon S3 or Hadoop Distributed File System (HDFS), which facilitate horizontal scalability and enable the storage of petabytes of data at a comparatively low expense.

A key characteristic of the storage layer is its capacity to accommodate data in diverse formats and optimization tiers, contingent upon querying and processing requirements. Formats like Parquet and ORC are frequently utilized for structured and semi-structured data due to their efficient compression and improved performance for analytical queries. These formats provide partitioning, permitting quick access to particular subsets of data without necessitating the processing of the full repository. Optimized formats and compression techniques enable the Data Lake to effectively facilitate both historical data analysis and real-time processing [14].

4.3. Data Processing Layer

The processing layer is tasked with transforming, cleansing, and structuring data within the Data Lake to facilitate analysis, reporting, and business activities. This layer may encompass pretreatment operations that structure data for subsequent analysis, along with data pipelines that execute intricate transformations, including aggregations, calculations, and data enrichment.

Data processing within the Data Lake may be conducted in either batch mode or real-time, contingent upon the organization's requirements. Batch processing is advantageous for extensive transformations, including the generation of aggregates or the application of machine learning models. Apache Spark and similar tools are extensively utilized for facilitating fast, large-scale data transformations. Conversely, real-time processing employs stream processing technologies like

Apache Flink or Apache Kafka Streams, enabling immediate data processing upon admission to satisfy low-latency demands for near real-time updates [3].

4.4. Data Consumption Layer

The consumption layer facilitates access to the Data Lake's data for diverse users and applications, enabling them to utilize and analyze the data based on particular requirements. This layer is crucial for enabling effective and safe access to Data Lake data for ad hoc queries, reporting, and operational applications that necessitate real-time data.

The Data Lake facilitates access to data for many users through analytical tools, business intelligence dashboards, machine learning applications, and external enterprise systems. Tools like Amazon Athena, which facilitates SQL queries directly on the Data Lake, and Power BI, utilized for creating interactive visual reports, are frequently employed to derive insights from stored data. Integration APIs can be created to allow external systems to access data in real time, utilizing streaming technologies for continuous and current availability [15].

4.5. Governance and Security in the Data Lake

An essential component of data lake architecture is the establishment of governance and security protocols that guarantee data integrity, compliance, and protection. Data Lake governance encompasses methods including data quality assurance, access auditing, authorization administration, and ongoing monitoring to guarantee ethical data utilization and adherence to rules such as GDPR and LGPD.

To guarantee data security, methods including encryption at rest and in transit, multi-factor authentication, and identity and access management (IAM) are employed. The Data Lake can be segmented into data zones (e.g., raw zone, processing zone, consumption zone), with data arranged based on its level of preparation and access requirements. These governance and security policies are crucial for risk mitigation and for ensuring that the Data Lake functions as a regulated and reliable environment for data storage and consumption.

In conclusion, Data Lake architecture offers a versatile and scalable framework for the storage and processing of substantial data volumes. By structuring data into intake, storage, processing, and consumption layers, the Data Lake enables enterprises to consolidate data from several sources into a unified repository—enhancing accessibility and supporting advanced analytics and real-time operations. Through effective governance and security measures, the data lake serves as a fundamental component of contemporary organizational data architecture, facilitating efficient, secure, and organized access to data for applications and users.

5. Using Data Lakes as a Centralized Integration Layer for Distributed and Monolithic Applications

The implementation of data lakes as a centralized integration layer efficiently re-

solves the issues of data integration in enterprise environments consisting of both monolithic systems and dispersed applications. Figure 5 demonstrates that the data lake functions as a central hub, assimilating data from various sources—comprising real-time streams and batch processes—and distributing it to multiple applications, while concurrently interfacing with foundational transactional databases. This separation from direct database access alleviates prevalent problems such as tight system coupling, undue strain on operational databases, and inconsistent data perspectives.

The data lake consolidates information into a cohesive and scalable architecture, allowing diverse applications to access and process data according to their specific needs—be it in real-time, near-real-time, or from historical datasets. This architecture improves data accessibility and consistency while providing enhanced flexibility and resilience, as applications are no longer directly reliant on transactional databases for every query.

Furthermore, the data lake supports a wide range of data formats and time frames, serving as a continuous and adaptable repository for both operational tasks and analytical applications (Figure 5). It establishes the technology framework for comprehensive data integration across the organization by delivering scalability, modularity, and governance features vital for facilitating contemporary data-driven strategies. The data lake functions not merely as a storage solution but as a strategic architectural element that optimizes data flow, diminishes operational complexity, and improves the agility of corporate systems.

The proposed centralized integration approach closely corresponds with the establishing Data Lakehouse paradigm [3] [8], which combines the scalability and adaptability of Data Lakes with the management, schema enforcement, and query optimization features typically linked to Data Warehouses. This hybrid model enhances the architectural value proposition by facilitating both sophisticated analytics and operational queries in a unified environment, while maintaining the separation from transactional databases emphasized in this study.

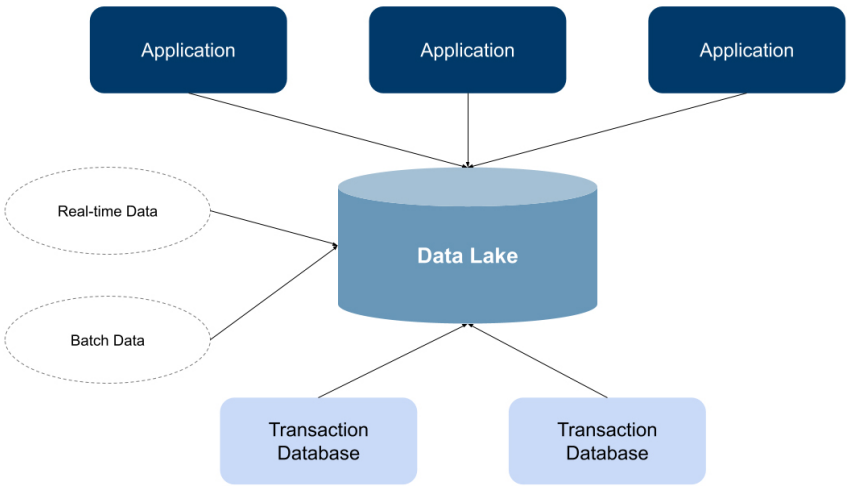

Figure 5. Data lake as a Centralized Integration Layer. Source: Author (2025).

5.1. Structuring the Data Lake for Centralized Integration

For a data lake to serve as an integration layer, it must be engineered to accommodate various temporalities and forms of data consumption. This architecture entails the establishment of discrete "zones" within the Data Lake, which categorize data according to its degree of transformation and preparedness for utilization. The primary zones typically utilized are:

- **Raw Zone**: Data is preserved in its original form, without any processing or alteration. This zone is appropriate for situations requiring raw data for comprehensive study or replication.
- **Staging Zone**: Data undergoes preliminary processing, which may encompass cleansing, normalization, and transformation to conform to the requisite format for utilization.
- **Consumption Zone**: Data is prepared for utilization by diverse applications. This zone organizes data for rapid and efficient queries, potentially utilizing formats like Parquet or ORC to enhance performance and minimize storage needs. [2]

Figure 6 illustrates how the data lake architecture can be organized with tiered storage systems, wherein frequently accessed data resides in high-performance storage tiers while infrequently accessed data is allocated to lower-cost tiers, as well as with logical data zones that represent various stages of data transformation and readiness for consumption. Data from various operational sources is ingested through ETL pipelines into a consolidated Raw Data zone, maintaining the original structure and format for traceability and compliance.

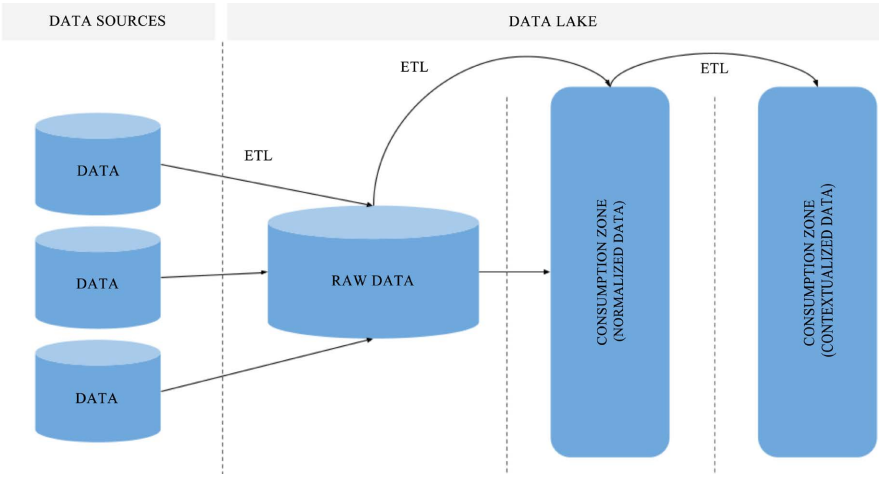

Figure 6. Data structuring and storage in Data Lakes. Source: Author (2025).

Data is subsequently processed and refined through successive ETL stages into designated Consumption Zones, typically comprising at least two layers: a Normalized Data Zone, where data from multiple sources is harmonized into a common schema, and a Contextualized Data Zone, where domain-specific business logic is applied to generate datasets suitable for analytical or operational use. This

progressive structuring facilitates cost and performance optimization via storage tiering and enhances data usability by enabling various applications to access data at the most appropriate level of abstraction—be it raw, standardized, or fully contextualized. This methodology improves governance, query efficiency, and scalability throughout the data lifecycle.

5.2. Data Consumption Mechanisms

Contemporary data architectures facilitate data consumption via diverse mechanisms that accommodate various application requirements and operating timelines (Figure 7). Contemporary applications require access to data in real time, at predetermined intervals, or from historical archives, contingent upon their functionality, latency tolerance, and processing objectives.

These consuming patterns—real-time analytics, periodic batch processing, and retrospective analysis—place varying demands on the foundational data infrastructure. Thus, each type of data consumption necessitates specific technologies, formats, and procedures to guarantee quick data retrieval, as well as its relevance, correctness, and timeliness for the application utilizing it.

For instance, real-time data streams may employ event-driven architectures and in-memory processing, whereas batch processes can utilize scheduled ETL pipelines and data warehousing methodologies. Conversely, historical data consumption frequently necessitates improved storage formats and indexing algorithms for targeted access and sustained efficiency.

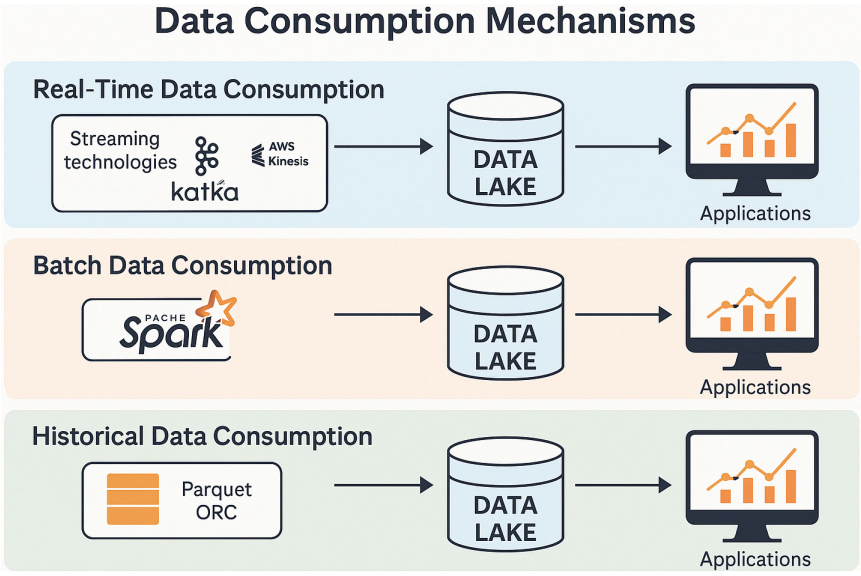

Figure 7. Data consumption mechanisms. Source: Author (2025).

Figure 7 demonstrates that comprehending and applying the suitable method for each consumption model is crucial for fulfilling performance requirements, preserving data integrity, and facilitating business-driven data use cases.

- Real-Time Data Utilization: For data requiring immediate access, the Data

Lake can utilize streaming technologies such as Apache Kafka and AWS Kinesis. These tools facilitate the processing and consumption of data concurrently with its ingestion, satisfying low-latency requirements. Applications necessitating current data—such as monitoring systems and real-time business intelligence dashboards—can interface with the Data Lake via data streams, thereby circumventing the burden of directly querying transactional databases [4].

- Batch Data Consumption (Periodically Processed Data): Batch processing is suitable for data that can be processed at regular intervals. In this strategy, data is periodically taken from the Data Lake and converted as required. Apache Spark and similar tools are extensively utilized for processing suited for daily, weekly, or monthly reports, as well as for historical analysis that do not necessitate real-time data. This consumption approach alleviates strain on operational systems and facilitates the efficient processing of substantial data quantities [3].
- Historical and Aggregated Data Utilization: The Data Lake retains past data in query-optimized formats, including Parquet and ORC. Analytical and visualization tools, such as Amazon Athena, may directly access this information, enabling the execution of SQL queries on data stored in the Data Lake. This form of consumption is optimal for longitudinal analysis, machine learning, and business intelligence, where a comprehensive history record is essential for discerning trends and patterns.

5.3. Advantages of Using the Data Lake as a Centralized Layer

The methodology presented in this paper provides substantial advantages for intricate corporate environments, frequently comprising a combination of monolithic and distributed applications. A key benefit is the alleviation of strain on transactional databases. Centralizing data consumption in the Data Lake alleviates excessive pressure on operational databases, enabling them to concentrate on essential transactional functions. This enhances the efficiency of production systems and mitigates the danger of deterioration during peak access intervals.

A significant advantage is the flexibility and scalability provided by utilizing the data lake as the integration layer. This architecture enables applications to expand more efficiently, as data is stored and retrieved independently of the operational systems. Applications can utilize data as required without impacting transactional systems, hence offering enhanced flexibility to adjust infrastructure to evolving business requirements.

The concept endorses centralized governance, hence facilitating the implementation of security, compliance, and governance regulations. Access rights, data quality standards, and audit processes can be administered cohesively within the data lake. This guarantees that all applications adhere to uniform standards and promote conformity with legislation such as GDPR.

Furthermore, the architecture facilitates efficient querying and storing. The data lake facilitates expedited queries and enhanced data storage efficiency

through the utilization of optimized data formats and tiered storage methodologies. This is especially crucial for historical data, which can be compressed and partitioned for targeted access, enhancing performance and minimizing storage expenses.

Ultimately, the architecture facilitates less connection among applications. Facilitating data access via the data lake, rather than through direct system interactions, reduces service dependencies. This streamlines maintenance and upgrades, removes the necessity for managing direct system-to-system connections, and enhances integration with new services, leading to a more flexible and resilient system design.

The centralized model shown here differs from the decentralized Data Mesh design [8], which allocates data ownership and governance among domain-specific teams. Data Mesh provides organizational scalability; nonetheless, its application necessitates substantial cultural and procedural transformations, potentially heightening integration difficulty. In situations when cohesive governance, uniform data quality, and efficient operational integration are essential, a centralized Data Lake strategy may prove more beneficial.

5.4. Challenges and Considerations

The proposed approach presents several benefits; however, important issues must be confronted when establishing a data lake as a centralized integration layer.

The main issue is the complexity of implementation. Implementing a data lake architecture requires a meticulously organized and resilient infrastructure, bolstered by suitable tools for data ingestion, storage, and processing. Alongside the technological elements, the implementation of governance, security, and data quality regulations demands a significant allocation of time, effort, and resources. In the absence of a definitive plan and developed operational processes, the data lake is at risk of devolving into a chaotic data swamp.

A significant concern is latency management, especially in real-time data contexts. Guaranteeing the ingestion, processing, and access of data to consuming applications with low latency is complex. While streaming technologies like Apache Kafka, Kinesis, or Flink mitigate latency, the whole architecture must be carefully designed to accommodate the necessary data amounts and update frequencies for various business applications. Neglecting to do so may jeopardize performance and responsiveness.

Ensuring data consistency and quality is crucial for the Data Lake to function as a reliable integration layer. This entails executing rigorous data validation, cleansing, and transformation procedures, in addition to instituting ongoing monitoring to ensure data accuracy, completeness, and timeliness. If the integrity of the ingested data is compromised, consuming applications may obtain obsolete or erroneous information, hence eroding faith in the platform.

Notwithstanding these obstacles, the integration of adaptable data storage and utilization, centralized governance, and diminished system coupling establishes

the data lake as a fundamental component of contemporary data architecture. When effectively executed, it allows enterprises to attain scalable and efficient data integration, facilitating both routine operations and strategic decision-making processes.

It is important to emphasize that this work is predominantly conceptual and lacks empirical performance assessments. Subsequent research will include benchmark testing and case studies that compare the proposed architecture with conventional integration approaches, measuring advantages in scalability, latency, and governance [3] [8]. These evaluations will substantiate the theoretical benefits outlined and offer tangible direction for implementation across various organizational settings.

6. Conclusions and Final Considerations

Implementing a data lake as a centralized integration layer provides significant benefits for enterprise IT architectures, especially in complex contexts where monolithic and distributed systems coexist. This method resolves numerous conventional architectural issues, including transactional database overload, restricted scalability, fragmented governance, and excessive system coupling, by unifying data consumption and organizing information for enhanced storage and accessibility. The outcome is a more robust, scalable, and flexible infrastructure that facilitates operational efficiency and strategic decision-making.

A primary advantage is the alleviation of strain on transactional databases. In legacy systems, these databases frequently manage both critical business functions and the requirements of analytics and reporting, resulting in performance deterioration. The Data Lake alleviates these secondary demands, enabling transactional systems to concentrate exclusively on mission-critical processes. This decoupling enhances system responsiveness, mitigates bottleneck risks at peak times, and fosters improved overall stability and user satisfaction.

Scalability and adaptability are significantly improved. The centralized paradigm allows applications to access common datasets without direct interaction with operational systems, facilitating independent service scalability. This is particularly advantageous in microservices architectures, where services can develop and scale independently. Technologies like Amazon S3 facilitate horizontal scaling, enabling the architecture to manage escalating data volumes without compromising performance.

The Data Lake simultaneously promotes centralized control and security. Organizations can systematically monitor data usage across all applications with unified access controls, audit logs, and compliance enforcement in accordance with requirements such as GDPR and LGPD. Organizing the lake into zones—raw, processing, and consumption—enhances varied access permissions according to data sensitivity and readiness, so assuring secure and traceable data management.

Efficiency in storage and querying is attained by the utilization of optimized file formats like Parquet and ORC, which provide high compression ratios and rapid query performance. These formats are especially advantageous for historical data,

minimizing storage expenses while preserving performance. Partitioning and metadata indexing facilitate selective and efficient data access, accommodating large-scale analytical workloads [16].

A significant benefit is the decrease in application coupling. Instead of depending on direct, point-to-point interfaces, applications obtain data from the Data Lake, allowing them to function autonomously. This autonomy streamlines development, deployment, and maintenance, while enhancing system resilience—failures in one system are less likely to impact others.

The architecture of the data lake accommodates various data temporalities, such as real-time, batch, and historical data. This adaptability enables various applications to utilize data in accordance with their distinct needs. Real-time streams facilitate operational monitoring, but historical datasets enhance business intelligence tools and machine learning models for strategic analysis and forecasting.

In this context, data lakes facilitate advanced analytics and machine learning. The Data Lake functions as a consolidated repository of clean, preprocessed, and systematically organized data, thereby mitigating numerous complications associated with data preparation. Analysts and data scientists obtain effortless access to varied datasets, facilitating expedited modeling and the automation of machine learning processes.

The Data Lake functions not merely as a storage repository but as a fundamental integration platform, augmenting scalability, governance, performance, and analytical capacity. It streamlines data access, diminishes reliance on operational systems, and enables a modular, future-ready design tailored to the requirements of contemporary organizations.

However, effective adoption necessitates surmounting implementation obstacles. This encompasses the necessity for resilient infrastructure, advanced governance mechanisms, and architectural frameworks that sustain performance under substantial data volumes. Continuous management of real-time integration, data quality assurance, and security is essential for maintaining long-term value.

Anticipating future advancements—such as AI-driven governance automation, ultra-low-latency architectures, and interaction with emerging technologies like IoT and blockchain—can further enhance the Data Lake model. These improvements are poised to enhance its function as the foundation of enterprise data ecosystems.

In conclusion, data lakes constitute a flexible and robust architectural option for modern companies. They consolidate governance, enhance storage and accessibility, and accommodate a wide array of data applications, ranging from operational dashboards to predictive analytics. The data lake is not merely a transient trend; it is establishing itself as a fundamental element in the contemporary data ecosystem, empowering enterprises to transform raw data into strategic assets and to develop scalable, intelligent, and resilient digital operations.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Miloslavskaya, N. and Tolstoy, A. (2016) Big Data, Fast Data and Data Lake Concepts. *Proceedings of the 7th International Conference on Dependable Systems, Services and Technologies* (*DESSERT*), Kiev, May 2016, 237-241.

[2] A.S. Foundation (2018) Apache Parquet. https://parquet.apache.org/

[3] Harby, A.A. and Zulkernine, F. (2025) Data Lakehouse: A Survey and Experimental Study. *Information Systems*, **127**, Article 102460. https://doi.org/10.1016/j.is.2024.102460

[4] Kreps, J., Narkhede, N. and Rao, J. (2011) Kafka: A Distributed Messaging System for Log Processing. *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, Athens, June 2011, 1-7.

[5] Richardson, C. (2018) Microservices Patterns: With Examples in Java. Manning Publications.

[6] Stonebraker, M., Madden, S., Abadi, D.J., Harizopoulos, S., Hachem, N. and Helland, P. (2007) The End of an Architectural Era (It's Time for a Complete Rewrite). P*roceedings of the 33rd International Conference on Very Large Data Bases* (*VLDB*), Vienna, 23-27 September 2007, 1150-1160.

[7] Fowler, M. (2015) Microservices. https://martinfowler.com/microservices/

[8] Gartner (2022) Data Lake Architecture and Use Cases. Document ID: 3980938. https://www.gartner.com/en/documents/3980938

[9] Hohpe, G. and Woolf, B. (2003) Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley.

[10] Erl, T. (2005) Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall.

[11] Papazoglou, M.P. (2007) Web Services: Principles and Technology. Pearson Education.

[12] Newman, S. (2015) Building Microservices: Designing Fine-Grained Systems. O'Reilly.

[13] Chappell, D.A. (2004) Enterprise Service Bus. O'Reilly.

[14] Alrehamy, H. and Walker, C. (2015) Personal Data Lake with Data Gravity Pull. *Proceedings of the 5th IEEE International Conference on Big Data and Cloud Computing*, Dalian, 26-28 August 2015, 160-167.

[15] Kimball, R. and Ross, M. (2013) The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. 3rd Edition, Wiley.

[16] Hai, R., Koutras, C., Quix, C. and Jarke, M. (2023) Data Lakes: A Survey of Functions and Systems. *IEEE Transactions on Knowledge and Data Engineering*, **35**, 12571-12590. https://doi.org/10.1109/TKDE.2023.3270101