Data Lake Architecture -

A Comprehensive Design Document

Medical Data Processing Company

# Tracker

## Revision, Sign off Sheet and Key Contacts

## Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Change Reference |
| 22-02-2023 | Jahid Razan | 1.0 | Initial draft |

## Reviewers / Approval

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Version Approved | Position | Date |
| FirstName LastName | 1.0 | Udacity Reviewer  Enterprise Data Lake Architect |  |

## Key Contacts

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Role | Team | email |
| Jahid Razan | Data Architect | Medical Data Processing | Jahid.razan13@gmail.com |

# Purpose

## 1.1 What does it contain ?

### This document contains the business background of the Medical Data Processing Company.

### It describes the challenges of the existing architecture and outlines a data lake solution that can solve the existing challenges of the infrastructure and use cases.

## 1.2 Why the Document was Created:

### This document aims to accomplish the following objectives:

### Describe the design components needed to meet existing and future challenges.

### Create a shared understanding among the CTO and the engineering design team.

### Facilitate informed and effective decision-making in technology selection for stakeholders across the organization.

## 1.3 Target Audience

### The target audience for this document includes the CTO and engineering team who are responsible for the company's technical infrastructure. This document has been created to provide a detailed design plan that will help solve the company's challenges and improve uptime, reduce latency, and increase business agility.

## 1.4 Scope of the Document

The scope of the document includes the following-

### The document lists the design principle to meet the technical and business requirements of the company.

### It lists the current infrastructure and data volume, business and technical requirements to of the company for the proposed data lake architecture.

### This document proposes a data infrastructure, the rationale behind the selection of different tools for ingestion, storage, processing, and serving layers, as well as the potential limitations and strategies to overcome them.

Following items are out of scope for the document -

### The configuration and detailed step-by-step configuration of tools in various layers performance monitoring and optimization, business logic, metadata management, and data modeling based on business cases, are not within the scope of this document.

* The migration of data from the existing infrastructure and business case implementation strategies are not within the scope of this document.
* Item related to implementation strategy and details such as cost, capacity planning are not within the scope of this document.

# Requirements

Medical Data Processing Company of San Francisco specializes in processing various types of EMR (Electronic Medical Records) and providing real-time insights to various medical facilities. The company has approximately 1,100 customers, and its solution is used by approximately 8,000 individual medical care facilities. Currently, the company hosts over 8TB of data in SQL Server.

The company has experienced hypergrowth over the past three years. As the data volume continues to increase, the existing single-node SQL Server is unable to scale, resulting in slow ETL processes and SQL reporting queries due to the increased data volumes. The ad-hoc solutions require maintaining multiple versions of the same data, which is a waste of already scarce memory resources. Tracking the latest version has become a complex, error-prone process and has resulted in data silos within the company. The existing backup and recovery procedure is extremely fragile, with a single point of failure. In the event of a network failure, there is no way to avoid long downtime, resulting in risks and poor customer experience.

The CTO wants to build additional capabilities to solve this problem. He envisions a solution that can solve the problem of storage, downtime and . He also envisions facilitating machine learning models for predictive analysis and near-real-time dashboards containing patient data for each facility without the need to move data from one system to another.

The company profile and problem statement document list the existing data architecture, technical environment, data volume, and technical and business requirements.

## 2.1 Existing Technical Environment

Table 1- lists the components of the existing data architecture of the company-

|  |  |
| --- | --- |
| Component | Description |
| Master SQL DB Server | Single node SQL server hosting critical customer data |
| Stage SQL DB Server | * 64 core vCPU * 512 GB RAM * 12 TB disk space (70% full, ~8.4 TB) * 70+ ETL jobs running to manage over 100 tables |
| Data ingestion server | 3 smaller server for FTP Server, data and API extract agents |
| Web and Application server | Multiple server with 32 GB RAM Each, 16 core vCPU |

## 2.2 Current Data Volume

Table 2- lists the data volume of the company-

|  |  |
| --- | --- |
| Parameter | Value |
| Data Source | 8K+ facilities |
| Zip files size (99% of cases) | 20 KB to 1.5 MB |
| Edge cases | File size to 40 MB |
| File format types after unzipping | CSV, TXT, XML records |
| XML zip files | 20-300 individual XML files |
| Average zip files per day | 77,000 |
| Average data files per day | 15,000,000 |
| Average zip files per hour | 3,500 |
| Average data files per hour | 700,000 |
| Data Volume Growth rate (YoY) | 15-20% |

## 2.3 Business Requirements

## Table 3- lists the business requirements of the enterprise for the proposed data lake-

|  |  |
| --- | --- |
| Requirement | Description |
| Improve system uptime | Increase availability and stability of the overall system |
| Reduce SQL query and report latency | Optimize database performance and improve query processing speed. |
| Reliability and fault tolerance | Ensure the system is resilient to failures and can recover quickly in the event of a failure. |
| Scalability | Ability to scale the system as data volume and velocity increases. |
| Speed of innovation and agility | Increase the speed of innovation and agility in the organization through automation and ability to experiment. |
| Open source tools | Utilize open source tools to avoid vendor lock-in and ensure flexibility. |
| Metadata driven design | Standardize the ETL process and avoid building custom scripts to process each type of data source. |
| Centralized data access | Store all enterprise data in a central location and enable easy access. |

## 2.4 Technical Requirements

Table 4- lists the technical requirements of the enterprise for the proposed data lake-

|  |  |
| --- | --- |
| Requirement | Description |
| Real-time data processing | Ability to process incoming files on the fly (instead of nightly batch loads today) |
| Separation of metadata, data and compute | Separate the metadata, data and compute/processing layers |
| Unlimited historical data storage | Ability to keep unlimited historical data |
| Scale up processing speed | Ability to increase processing speed as data volume and velocity increases |
| High availability | System should sustain small number of individual node failures without any downtime |
| Change Data Capture (CDC), UPSERT support | Ability to perform CDC and UPSERT on a certain number of tables |
| Multiple use cases from same dataset | Ability to drive multiple use cases from the same dataset, without the need to move or extract the data |
| Integration with ML frameworks | Ability to integrate with different ML frameworks such as TensorFlow |
| Dashboard and report generation | Ability to create dashboards using tools such as PowerBI, Tableau, or Microstrategy, and generate daily/weekly reports |
| Ad-hoc data analytics | Ad-hoc data analytics, interactive querying capability using SQL |

# Data Lake Architecture design principles

The mentioned business case identifies several crucial design principles that must be implemented in order to achieve the technical and business objectives:

* **Scalability:** Currently, the SQL database of the company lacks the capacity to store the valuable historical data, as well as the forecasted 15-20% year-over-year growth in data volume. To address this, the system needs to be scalable.
* **Fault Tolerance**: The industry in which the company operates is highly sensitive, and decisions related to life and death may be affected by system availability. The existing system's restoration mechanism takes hours to restore when it breaks, which is unacceptable. Therefore, the future system needs to be reliable and fault-tolerant, which can be achieved through an architecture with redundancy, smart backup mechanisms, and no single point of failure. The backup and recovery processes must be much stronger.
* **Reduced Latency:** Due to insufficient capacity, the company currently exports required data on a nightly basis to separate servers for analytical purposes. This has created many data silos, and each department has found a way that suits them, resulting in the need to keep track of the most up-to-date data location for hundreds of tables, which is error-prone. To perform near-real-time dashboard and analytics and establish a single source of truth without confusion or the need to move or extract the data, the company needs a low-latency system.

In addition, to avoid potential vendor lock-in, the data architecture will make use of open-source tools with the above-mentioned characteristics as much as possible.

# Assumptions

## 4.1 Missing information on policy level

Table 5 – summarizes the policy level information that are missing in the existing document:

|  |  |  |
| --- | --- | --- |
| Policy and process | Currently Missing | Assumptions |
| Data privacy and security requirements | Details of how the data privacy and security requirements, including any relevant regulations, such as HIPAA or GDPR, and any encryption or access control measures that need to be implemented. This has the potential risk of failing to meet local and federal regulations. | There are stringent data privacy and security requirements in place, and all sensitive data must be encrypted and access-controlled. |
| Data retention policies | Details of the current data retention policies, including how long data needs to be stored, how often it needs to be purged, and any legal or compliance. requirements that need to be met. | The data retention policies require storing all data indefinitely, or for a long period of time. |
| Disaster recovery plan | Details of the current disaster recovery plan, including backup and restore processes, recovery time objectives (RTO), recovery point objectives (RPO), and any other relevant information | No rapid backup and recovery plan in place. |
| Current ETL process flow and scripts | Details of ETL processes and scripts, how they handle each type of file, and the processing logic. | Existing ETL processes are complex and time-consuming due to the custom-designed scripts and the large volume of data being processed. ELT processes for the data lake will be standardized. |

## 4.2 Missing implementation details

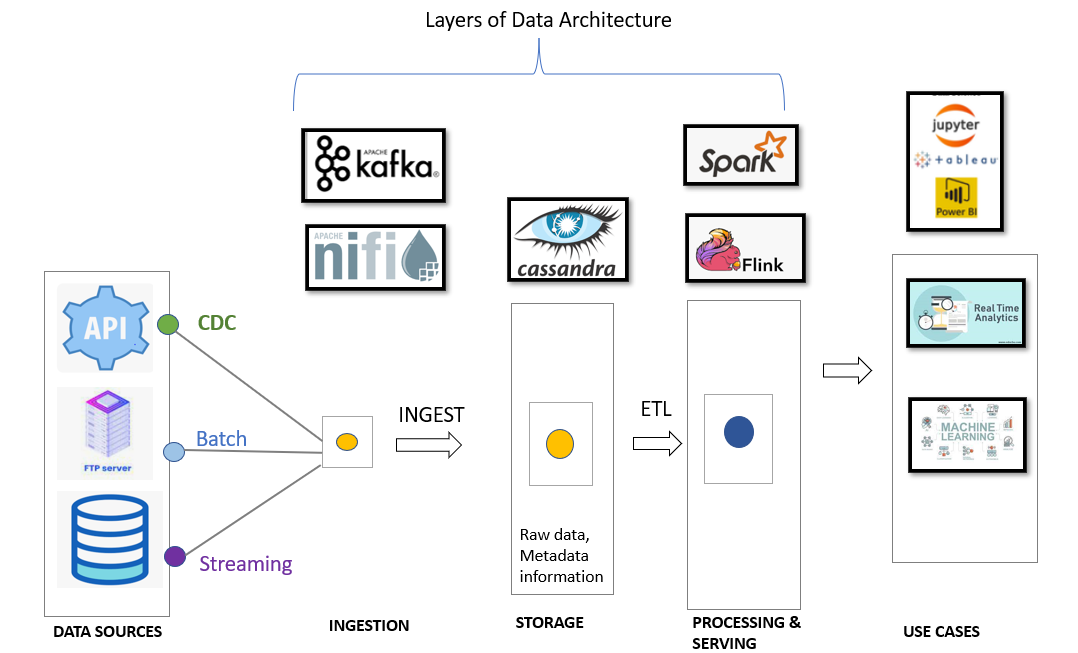
Table 6 – lists the missing items in the existing document

|  |  |  |
| --- | --- | --- |
| Business/Technical Requirement | Currently Missing | Assumptions |
| Improve uptime of overall system | What causes the system downtime of the existing system? | Current downtime is caused by hardware failure/limitations and inability to scale |
| System should be reliable and fault tolerant | What is the acceptable level of downtime for maintenance and upgrades ? | The company requires a system with at least 99.9% uptime, allowing for up to 8 hours of downtime per year. |
| System should sustain small number of individual node failures without any downtime | What is the acceptable number of node failures? | The future architecture will be able to withstand 2 node failures without any downtime |
| Improve business agility and speed of innovation through automation and ability to experiment with new frameworks | What are the specific automation and innovation goals? | The company wants to fully automate ETL processes to eliminate all manual effort required for data management. |
| Embrace open source tools, avoid proprietary solutions which can lead to vendor lock-in | What are the open source tools the company is interested in ? | The company is open to using a variety of open source tools and willing to use them in combination to overcome challenges that a single open-source tool might offer. |
| Centrally store all of the enterprise data and enable easy access | What are requirements for different user groups? | The company requires role-based access control, with different levels of access for analysts, data scientists, and executive |
| Ability to perform change data capture (CDC), UPSERT support on a certain number of tables | What are the specific tables and fields requiring CDC and UPSERT support ? | The company requires CDC and UPSERT support on all tables and fields containing patient data |

Without knowing the specific details of items like specified tables for CDC and UPSERT operation, duration of data retention, uptime requirement – it is not sure to pinpoint whether the requirements have been met. As a starting point, the assumptions have been made which needs to be validated with the key stakeholders.

# Data Lake Architecture for Medical Data Processing Company

Figure 1: Shows the proposed data architecture with different layers



Inspiration for Data Architecture Diagram: https://hevodata.com/learn/data-ingestion/

The data architecture consists of-

* Ingestion layer
* Storage Layer
* Processing Layer
* Serving Layer

## 5.1 Ingestion Layer

Table 7 – summarizes the tools and criteria considered for the ingestion layer:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Tool | Scalability | Fault Tolerance | Latency | Open Source | Potential Risks of Vendor Lock-in | Used in real time data analysis and processing in Healthcare Applications |
| Sqoop | Limited | Limited | High | Yes | No | No |
| Apache Kafka | High | High | Low | Yes | Yes | Yes |
| Amazon Kinesis | High | High | Low | No | Yes | Yes |
| Google Pub/Sub | High | High | Low | No | No | Yes |
| Apache nifi | High | High | Low | Yes | No | Yes |

Apache Kafka and Apache nifi are the two tools with strong options for healthcare applications. Both the tools are highly scalable, fault-tolerant, and low latency, making them suitable for real-time data processing and analysis.

Kafka is better suited for use cases that require high-throughput, low-latency data ingestion and processing. NiFi is better suited for use cases that require complex data routing, transformation, and integration with multiple systems. Using them in conjunction can help tackling variety of cases. The use cases for both the tools for the company are listed below:

Table 8 – compares the Kafka and Nifi and recommends one for ingesting data from the existing data sources-

|  |  |  |  |
| --- | --- | --- | --- |
| Data Source | Kafka | NiFi | Reasoning |
| SQL DB server | **X** |  | Kafka can act as a high-throughput, distributed messaging system that can handle large volumes of data in real-time, making it a good choice for streaming data from SQL DB server. |
| FTP Server |  | **X** | Both Kafka and NiFi can handle data ingestion from FTP servers, but NiFi has built-in capabilities to directly connect to FTP servers and pull data, making it an easier choice. |
| Data and API extract agents |  | **X** | Both Kafka and NiFi can handle data and API extract agents, but NiFi has built-in processors to handle API data ingestion and can connect to different endpoints using its numerous connectors, making it a better choice. |
| Web and application servers |  | **X** | Both Kafka and NiFi can handle data ingestion from web and application servers, but NiFi's built-in web interface and processors that can parse and extract data from web servers make it a better choice. |

## 5.2 Storage Layer

Table 9 - summarizes the tools and criteria considered for the storage layer:

|  |  |  |  |
| --- | --- | --- | --- |
| Criteria | HDFS | Apache Cassandra | Apache HBase |
| Capability to Store vast amounts of data | Can store large datasets | Yes | Yes |
| Ability to handle CSV, TXT and XML records | Yes | Yes | Yes |
| Capability to handle 20% YOY data growth rate | Yes, can scale horizontally by adding more nodes. | Yes, can add more nodes to scale. | Yes, can scale horizontally. |
| Supports Backup and recovery strategies | Yes | Yes | Yes |
| Ability to store custom metadata store | Yes | Yes | Yes |
| Supports authentication and access control. | Yes | Yes | Yes |
| Possible vendor lock-in | No, open source technology. | No, open source technology. | No, open source technology. |
| Ability to perform Change Data Capture (CDC) and UPSERT on a certain number of tables | No | Yes | Yes |

Based on the criteria, Apache Cassandra can be selected as it is one the best tools for storing and processing large amount of data and can handle variety of data format such as CSV, TXT and XML records. **It can handle 20% YoY growth** and has the security feature. As it is an open source technology, it reduces the risk of possible vendor lock-in.

#### 5.2.A Limitations of Apache Cassandra are:

1. It lacks a unified query language to manage data across different data sources.
2. The tools requires manual configuration and optimization can be hard to obtain.
3. It has limited support for ACID transactions and can make it hard to for application with strict data consistency.
4. It can be challenging for somebody who has little to no background in distributed system to master.
5. To address these shortcomings, third-party tools like **Apache Spark** in the processing layer can be used. It can handle complex queries on data stored in Cassandra by connecting to Cassandra using the Cassandra connector and performing queries using the Spark SQL API. Spark's flexible APIs and data processing capabilities can help overcome some of the data modeling challenges. Spark can provide additional transaction support on top of Cassandra's storage layer by implementing ACID transactions using Spark's DataFrame API.

#### 5.2.B Strategy to store vast amount of data:

* **Data Modeling:** To get the best out of Apache Cassandra it is important to model data based on requirements and use cases. This requires alignment with the users.
* **Partitioning:** For data partitioning Cassandra uses partition key. This can be achieved by using a unique ID such as Universal Unique Identifier (UUID) .
* **Compression:** Cassandra has a built in algorithm that help reducing the data size by compression. To store a vast amount data it is recommended to compress file size larger than 64 kb.
* **Performance Monitor:** Cassandra provides metrics for performance monitoring. Tools like Grafana can be used to visualize performance

#### 5.2.C Data Back Up and Recovery

Apache Cassandra has the following two backup and recovery strategies that can be used:

* **Incremental backups:** In incremental backup strategy, takes a backup for the data that has changed since the last backup. This is faster and require less storage than full backup. Cassandra provide the utility tool `nodetool` for incremental backup. This can be scheduled at a regular back up.
* **Point in time recovery (PITR):** It can help recover data from accidental deletions, or other data loss scenarios. Cassandra has `sstableloader` that you can use to recover data using PITR.

Cassandra also provides snapshot recovery that takes a snapshot of the entire dataset at a particular point in time. However, this is not recommended for regular backup as it takes larger space and longer time to restore compared to the incremental and PITR

#### 5.2.D Data Storage Format

Apache Cassandra is best suited for storing denormalized data. Because the tool is designed for high write throughput and scalability, and denormalized data models can help optimize these capabilities.

#### 5.2.E Custom Metadata:

Following metadata can be stored in Apache Cassandra-

* **Timestamps:** Cassandra has built in timestamp type to define timestamp.
* **Version Number:** It is possible to store version numbers of the data to track changes over time. Can be useful data quality monitoring, data synchronization of distributed system.
* **User-defined properties**: User ca define status such as ‘active’ or ‘inactive’ status,
* **Tags:** It is possible to use tag in cases like customer identity.
* **Audit Trail:** It is also possible to audit trail alongside track to see who made the changes in data, when. This can be useful for compliance and security purposes.

#### 5.2.E Secure data strategies

Important strategies to secure the stored data at Cassandra are –

* **Authentication and Authorization:**
  + Cassandra's built-in authentication and authorization mechanisms can be used to control access to the database and ensure that only authorized users are able to access data.
* **Encryption:**
  + Cassandra provides support for data encryption at rest using industry-standard encryption algorithms such as AES and RSA. This can be used to encrypt data stored on disk and prevent unauthorized access to sensitive data.
* **Network security:**
  + To ensure network security - it is important to properly configure network settings such as firewall rules and network access control lists (ACLs) to restrict access to the Cassandra cluster.

Third-party tools such DataStax Enterprise Security can also be used to provide additional authentication and authorization, for data encryption in transit using SSL/TLS encryption and to provide secure communication between nodes in the cluster.

## 5.3 Processing Layer

Table 10 - summarizes the tools and criteria considered for the processing layer-

|  |  |  |  |
| --- | --- | --- | --- |
| Criteria | Hadoop | Spark | Flink |
| Support Batch, Realtime, CDC data processing | Yes | Yes | Yes |
| Ad-hoc querying capabilities | Yes | Yes | Yes |
| Open source | Yes | Yes | Yes |
| Scalability | High | High | High |
| Fault Tolerance | High fault tolerance with replication and self-healing | High fault tolerance with RDDs and lineage tracking | High fault tolerance with stateful streaming and recovery mechanisms |
| Speed | Moderate | High | Very High |
| Application on health care technologies | Widely used in healthcare for processing large volumes of data | Widely used in healthcare for processing large volumes of data | Increasingly used in healthcare for processing large volumes of real-time data |

Apache Spark and Flink both can be used together as they can complement each other as data processing layer tools.

5.3.A Plan for scaling

Both frameworks are designed to process large amounts of data in parallel across a distributed cluster of machines. To maximize **performance and scalability**, a hybrid processing model can be considered that leverages the strengths of both frameworks.

* Apache Spark is well-suited for batch processing, interactive querying, and machine learning applications. It provides a high-level API and a rich set of libraries for processing data, including SQL queries, machine learning algorithms, graph processing, and streaming data processing.
* Flink is designed for stream processing and real-time data processing. It has a low-latency, high-throughput architecture that can process data in near real-time. It also supports batch processing and graph processing.
* It is important to implement a distributed architecture that can handle large volumes of data and high processing loads. This can be achieved by deploying multiple instances of Spark and Flink across a cluster of nodes, and using tools such as Apache Mesos or Apache Hadoop YARN to manage resource allocation and job scheduling.

#### 5.3.B How to satisfy different processing needs?

Apache Spark will be used for batch processing. For real time and change data capture- Flink will be used.

* **Batch Processing:** Apache Spark is more suitable for batch processing as it provides a rich set of APIs for large-scale batch data processing. It has optimized support for batch processing use cases, including ETL, data warehousing, and data integration.

Apache Flink also has support for batch processing, but it is more geared towards stream processing.

* **Real-time processing:** Apache Flink is better suited for real-time processing as it was designed with streaming data processing in mind. It offers low latency and high throughput processing capabilities and has built-in support for event-time processing and windowing. Apache Spark also supports real-time processing but spark's streaming performance may not be as good as Flink's in certain scenarios.
* **Change data capture (CDC) processing:** Apache Flink is more suitable for CDC processing as it provides native support for event-driven CDC use cases. Flink can consume change events from a variety of sources and process them in real-time, making it well suited for use cases such as data replication, data integration, and data synchronization. Apache Spark also supports CDC processing through third-party libraries, such as Debezium. However, the performance may not be as good as Flink's due to its focus on batch processing use cases.

#### 5.3.C How to enable ad-hoc querying capabilities?

Enabling ad-hoc querying capabilities using Apache Spark and Apache Flink typically involves using the **SQL interface** provided by both the platforms, Spark SQL and Flink SQL. These interfaces allow users to write SQL queries that can be executed on distributed datasets, without having to write complex code for data processing. Spark SQL is better suited for analyzing large datasets, while Flink SQL is better suited for real-time analysis of streaming data.

## 5.4 Serving Layer

#### 5.4.A What is a serving layer ?

Serving layer provides low-latency access to data for consumption by end-users or downstream systems. It is responsible for providing fast and reliable access to data, often through APIs or other query interfaces. The layer is a critical component of a data lake architecture, as it allows end-users to interact and utilize h the data stored in the lake in a fast and efficient manner and derive value from their data in real-time.

#### 5.4.A Types of stored data

**Apache Flink and Apache spark together** can be used as a serving layer for real-time analytics. Apache Spark and Apache Flink can be used together in the serving layer, depending on the use case and the specific requirements of the application. Both tools have strengths and weaknesses in different areas of data processing and can complement each other when used together.

|  |  |  |  |
| --- | --- | --- | --- |
| Types and formats of stored data | Apache Spark | Apache Flink | Better suited |
| Structured data (CSV, Parquet, Avro, etc.) | **X** | **X** | Apache Spark |
| Unstructured data (JSON, text, etc.) | **X** | **X** | Apache Spark |
| Predictive analytics | **X** | **X** | Apache Spark |
| Real-time monitoring |  | **X** | Apache Flink |
| Data visualization | **X** |  | Apache Spark |

#### 5.4.A How the data would be used ?

* Apache Spark is better suited for batch processing of structured and unstructured data, machine learning, and data visualization,
* Apache Flink is better suited for real-time processing of streaming data, especially for event-driven use cases that require low-latency processing and windowing.

# 8. Conclusion

Apache nifi, Kafka, Apache Cassandra, Apache spark, and Apache Flink, can work together in a data architecture that is scalable, fault-tolerant, and low-latency. Together, they can form a powerful data processing pipeline that can handle large volumes of data in real-time.

The potential challenges are :

* **Data consistency**: The tools have different approaches to data consistency, and ensuring that data is consistent across the entire pipeline can be a complex task.
* **Performance tuning:** Each component in the architecture requires its own performance tuning, and optimizing performance across the entire pipeline can be challenging.

* **Configuration**: Configuring Nifi and kafka to ingest data at high volumes, Cassandra to store large volumes of data, Apache Spark and Apache Flink to process data in real-time, and can require significant expertise.
* **Resource management:** Managing resources across the different components can be challenging. Flink and Cassandra both require significant amounts of memory and CPU resources, and ensuring that they have the necessary resources to run smoothly can be challenging, especially as the volume of data increases.

However, the benefits of a scalable, fault-tolerant, and low-latency data architecture can outweigh the challenges if done properly.

# 9. References

* Apache Kafka documentation: <https://kafka.apache.org/documentation/>
* Apache nifi documentation: [https://nifi .apache.org/docs.html](https://nifi.apache.org/docs.html)
* Apache Cassandra documentation:https://cassandra.apache.org/\_/cassandra-basics.html
* Apache HBASE reference guide: <https://hbase.apache.org/apache_hbase_reference_guide.pdf>
* Amazon Kinesis Documentation: <https://docs.aws.amazon.com/kinesis/index.html>
* Google pubsub Documentation: https://cloud.google.com/pubsub/docs
* Apache Spark Documentation: <https://spark.apache.org/documentation.html>
* [What is Apache Flink? Architecture, Use Cases, and Benefits](https://nexocode.com/blog/posts/what-is-apache-flink/)
* [Nifi System Administrator’s Guide](https://nifi.apache.org/docs/nifi-docs/html/administration-guide.html#clustering)
* [Data Lake Architecture: A Comprehensive Guide](file:///C:\Users\jahid.razan\Desktop\Udacity_Data_viz\Data_Architect_Nanodegree\Project_04\Submission_03\•%09https:\www.virtasant.com\blog\data-lake-architecture)
* [The Ultimate Guide to Data Architecture](https://www.integrate.io/blog/the-ultimate-guide-to-data-architecture/)
* [Top 9 Challenges of Big Data Architecture and its solutions | Overview](https://www.xenonstack.com/insights/challenges-of-big-data-architecture)
* [Challenges of Big Data Architecture](https://medium.com/@veeeresh423kumar/challenges-of-big-data-architecture-ef931f968c95)
* [Effective Data Architectures- Challenges and Solutions](https://blog.camelot-group.com/2022/05/effective-data-architectures-challenges-and-solutions/)