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
06/04/2020	Srijana Thapa	0.1	Initial draft

Reviewers / Approval

Name	Version Approved	Position	Date
Srijana Thapa	1.0	Udacity Reviewer Enterprise Data Lake Architect	

Key Contacts

Name	Role	Team	email
Srijana Thapa	Data Architect	Medical Data Processing	student@email.com

1. Purpose

The document outlines the architecture and design of the Data lake System for Medical Record Processing Company. The goal of this project is to redesign and improve the data layer- the ingestion, processing, storage and serving of data.

<What does the document contain?>

The document contains:

1. Requirements
2. Data Lake Architecture design principles
3. Assumptions
4. Data Lake Architecture for Medical Data Processing Company
5. Design Considerations and Rationale
6. Conclusion

The purpose of the document is to modify the existing technical environment and develop a Data lake Solution for the medical record processing company. It will guide the technical team on the Systems design and architecture to ensure it functions efficiently and scales seamlessly. The target audience includes:

1. Data engineers
2. Data Analysts
3. Project Manager
4. Technical directors
5. Enterprise architects
6. External partners

<u>In-scope</u> -

a. Create a scalable system
b. Centrally store all enterprise data and enable easy access
c. Improve uptime of the overall system
d. Ensure the system is available 24/7 and fault-tolerant

<u>Out of scope</u> –

a. Implementation of the data architecture
b. Providing continuous service to users
c. System reliability

2. Requirements

Medical data processing is facing problems with its current data management system due to rapid growth. The system can’t handle the increased data volume, which leads to slow processing, delays in reporting, and system crashes. The SQL server, the main database, is overwhelmed and can only process data at night. The company has no quick way to recover from system failures, causing long downtimes. There’s also an issue with having multiple copies of the same data, making it hard to track the most recent information. Moreover, the system lacks the ability to provide real-time analytics and support for machine learning.

To address these issues, a Data lake solution is proposed. The solution will provide scalable, reliable and efficient data storage and processing. Ensuring that the system can effectively handle both current and future data needs.

Existing Technical Environment

- 1 Master SQL DB Server
- 1 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
- 3 other smaller servers for Data Ingestion (FTP Server, data and API extract agents)
- Series of web and application servers (32 GB RAM Each, 16 core vCPU)

Current Data Volume

- Data coming from over 8K facilities
- 99% zip files size ranges from 20 KB to 1.5 MB
- Edge cases - some large zip files are as large as 40 MB
- Each zip files when unzipped will provide either CSV, TXT, XML records
- In case of XML zip files, each zip file can contain anywhere from 20-300 individual XML files, each XML file with one record
- **Average zip files per day:** 77,000
- **Average data files per day:** 15,000,000
- **Average zip files per hour:** 3500
- **Average data files per hour:** 700,000

Business Requirements

- Improve uptime of overall system
- Reduce latency of SQL queries and reports
- System should be reliable and fault tolerant
- Architecture should scale as data volume and velocity increases
- Improve business agility and speed of innovation through automation and ability to experiment with new frameworks
- Embrace open source tools, avoid proprietary solutions which can lead to vendor lock-in
- Metadata driven design - a set of common scripts should be used to process different types of incoming data sets rather than building custom scripts to process each type of data source.

  Centrally store all of the enterprise data and enable easy access

Technical Requirements

- Ability to process incoming files on the fly (instead of nightly batch loads today)
- Separate the metadata, data and compute/processing layers
- Ability to keep unlimited historical data
- Ability to scale up processing speed with increase in data volume
- System should sustain small number of individual node failures without any downtime

- Ability to perform change data capture (CDC), UPSERT support on a certain number of tables
- Ability to drive multiple use cases from same dataset, without the need to move the data or extract the data
    - Ability to integrate with different ML frameworks such as TensorFlow
    - Ability to create dashboards using tools such as PowerBI, Tableau, or Microstrategy
    - Generate daily, weekly, nightly reports using scripts or SQL
- Ad-hoc data analytics, interactive querying capability using SQL

As a fresher, my understanding of these requirements comes from a Udacity course and materials about big data systems and data architecture. These lessons have introduced me to key concepts like data ingestion, storage, processing frameworks and scalable data lake architecture. I learned about important tools like Apache Kafka, sqoop, spark, HDFS and YARN, which are essential for modern data solutions.

3. Data Lake Architecture design principles

In this project, we want to create a Data lake that is scalable, reliable and efficient.
First, we need to understand how much data we have now and how much we'll have in the future. We used HDFS (Hadoop Distributed file system) to store all the data in one place, with clear layers for how we bring in and store in data. This helps us plan how much storage we need and how to scale up when needed.
For ingestion the data, we used Apache Sqoop and Apache Kafka. Sqoop is good for batch processing and Kafka is good for real-time data. This gives us the flexibility to handle different kinds of data.
We keep storage and processing separate. We use HIVE for querying and managing metadata and Spark for processing data. The separation helps us choose the best tools for each task, making the system more efficient.
These principles ensure that our Data Lake can handle growing data, work with various tools, process data quickly, manage metadata well and use resources efficiently. This approach gives long term benefits for the Medical Data Processing Company.

4. Assumptions

- I assumed that the volume of medical data will keep growing at a rate of 15-20% every year. So, I designed the system to be scalable using HDFS to handle large data volumes.
- I assume that data will come from many different sources in different formats like CSV, TXT, and XML. That's why I included tools like Apache Sqoop, NiFi for batch data processing and Apache Kafka for real-time data streaming.

- I assumed it’s easier and more efficient to store all the data in one place. So, I used HDFS (Hadoop Distributed File System) to centralize the data for easier access and management.
- I assume the processing requirements would vary, needing tools for both batch and real-time processing. This is why I choose tools like Spark and HIVE for processing, as they provide flexibility and efficiency.

<u>Questions addressed while designing the architecture</u>

1. Which data formats of source data are to be ingested?
2. What size of the data to be in ingested?
3. What ingestion tools will we used?
4. How will we store the data?
5. Data growth in the coming days?
6. What storing tools will we use for our data?
7. What processing tool is the best?
8. Which orchestration tool will we use?

Missing in the problem statement – I assume the number of users based on the size of data, existing clients and predicted growth.

Potential risks that may be created now or in future based on these assumptions- The Company might need to buy new hardware for the Hadoop cluster.

5. Data Lake Architecture for Medical Data Processing Company

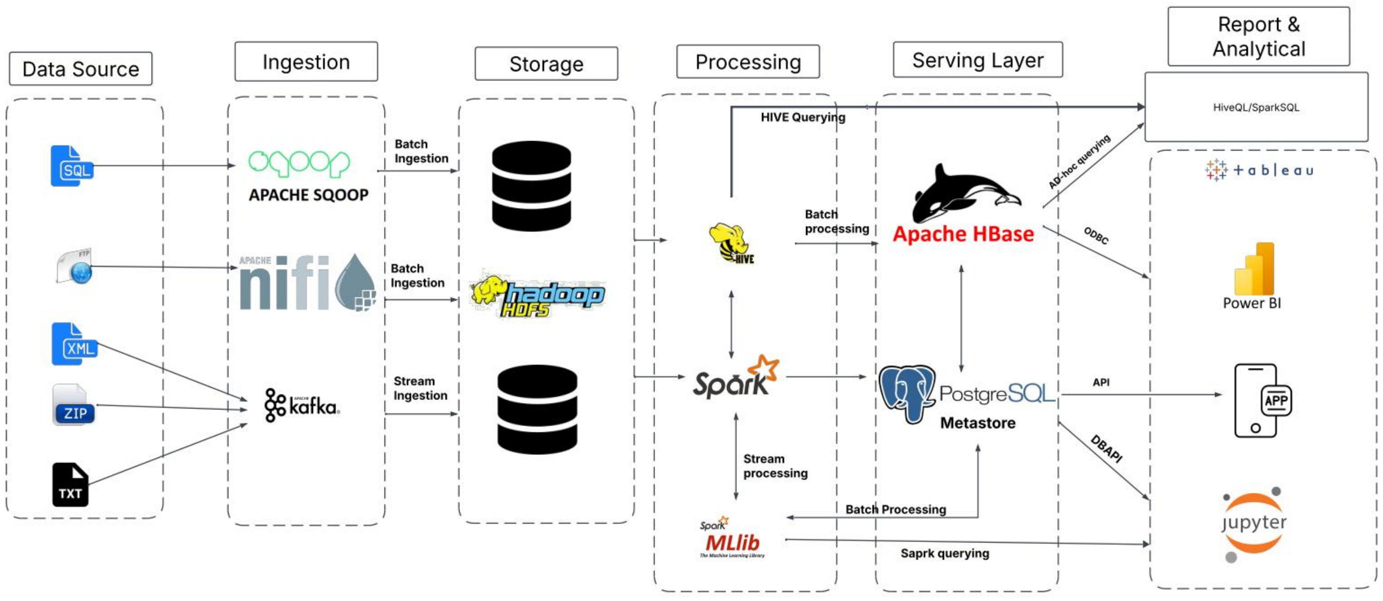

Figure: Data Lake Architecture

6. Design Considerations and Rationale

a. Ingestion Layer

We will use specialized tools to handle various formats like XML, TXT and CSV.

I ingest data coming from Databases, FTP servers, APIs are-
Databases: we will use Apache Sqoop to import data from databases like SQL server into HDFS. Apache Sqoop is good for moving large amount of data batches.
FTP servers: we will use Apache NiFi pr custom script to fetch and transfer files from FTP servers to the Data Lake. Apache NiFi helps in automating data flows and is flexible enough to handle different sources.
APIs: we will use Apache Kafka to handle real-time data streams from APIs and push the data into the Data Lake. Apache Kafka is great for real-time data because it can process high volumes of data with low latency.

The tools would be used : -
Apache Sqoop: It's efficient for batch data transfers from relational databases to HDFS. This is helpful when we need to move data at specific intervals.
Apache NiFi: It's a flexible data integration tools that supports various data sources and formats, including FTP servers. This makes it easy to automate the data flow from multiple sources.
Apache Kafka: It's ideal for real-time data ingestion and streaming from APIs, ensuring low latency and high throughput. This helps in processing data as it comes in.

The ingestion layer can scale by adding more nodes to handle increasing data volumes and sources. This is called horizontal scaling. Tools like Apache Kafka, Sqoop and Apache NiFi can process large amounts of data quickly and can easily grow to handle large data.

other tools considered
Apache Flume: It's use to collecting and moving log data. Why not chosen because it's less flexible than Apache NiFi for handling different data sources and formats.

b. Storage Layer

To store a large volume of data, we will use a System called HDFS (Hadoop Distributed File System) , which can handle large amounts of data.

Hadoop is open-source and easy to scale by adding more hardware, which will handle the 20% yearly data growth effectively.

To handle backup and recovery using Hadoop Distributed File System (HDFS), Schedule regular automates backups to ensure data is consistently backed up without manual intervention. We will store backups across multiple HDFS(Hadoop Distributed File

System) nodes to ensure that if one node fails, data can still be recovered from other nodes.

We will store custom metadata in Postgres database. The metadata will include information such as the file type (e.g., CSV, XML, TXT) creation date, data source (e.g., FTP server, API), owner, schema details (columns and data types) and versioning information (different versions of data files). This setup helps us organize and manage the data efficiently.

We will use the Parquet format for our data. It's really good for querying and save space because it compresses data well. This makes it perfect for handling large amounts of data quickly and efficiently.

<How do you plan to secure data (at a high-level)? Identify 2-3 techniques/tools/considerations>
<What other tools were considered? (3rd party tools, open source tools considered but did not make it to the architecture you are proposing). Are there other shortcomings to your selection of tools? If so what? Does the 3rd party tool solve that?>

Other tools
Another tool we can consider is Snowflake, a cloud data platform that handles ETL (Extract, Transform and Load) operations, data staging, processing and storage. Another option is the AWS S3 bucket, an object storage service used for storing and retrieving data. However, I did not choose these options because they require high maintenance costs. On the other hand, HDFS is an open source solution that offers the same results as AWS EMR without the high maintenance costs.

c. Processing Layer

We will process the data using batch and stream processing. CSV, XML, and text files will be processed in batches and loaded into HDFS or HBase. For SFT/SFTP and API data, we will used spark streaming for fast processing. HIVE will be used for aggregating, cleaning, and transforming data, which will then be stored in HBase. Sqoop will be used to transfer data between Hadoop and relational database. Once the stream is processed into required format the data can be accessed by SparkMlib for Machine learning.

The different processing needs Batch, Real-time, CDC :-
Batch processing- Apache NiFi and sqoop will handle batch processing by collecting and processing large volumes of data at scheduled times.
Real –time – Apache Kafka and Spark Streaming will manage real-time data as it arrives.
Change Data Capture (CDC) –Apache Kafka will be used to capture and track changes in data in real-time.

Ad-hoc querying will be enabled using Apache HIVE. These tools allow users to run SQL queries on large datasets for quick insights. Specifically, we will use HiveSQL.

The different tools are involved for processing:

- **Apache Nifi**: Data ingestion and batch processing.
- **Apache Kafka**: Real-time data streaming and processing, including CDC.
- **Spark Streaming**: Fast processing of streaming data.
- **Sqoop**: Batch processing and data transfer between Hadoop and relational databases.
- **Apache Hive**: Data warehousing and ad-hoc querying.
- **HBase**: Storing processed data for further analysis

Others tools

Other tools we considered for this project include Apache Pig and MapReduce for data processing. Apache Pig uses a scripting language called Pig Latin to write data transformations, but it is slower compared to modern tools. MapReduce processes large data sets in parallel, but it depends on disk for each iteration, which makes it slow. On the other hand, Apache Spark is much faster and more efficient for real-time queries because it reads and writes data in random access memory (RAM). Spark is 10-100 times faster than MapReduce. This combination of Spark and Hive ensures both high performance and ease of use for our data processing architecture.

The architecture can scale easily by adding more nodes to handle more data and processing demands. Tools like Apache Kafka and Apache NiFi can handle high volumes of data and can grow without much hassle. This ensures that our data processing is efficient, scalable, and versatile. We can query data stored in HDFS or Hive using HiveSQL. We can also query HBase using SQL. This setup makes accessing data flexible and efficient for different analytical needs.

d. Serving Layer

The serving layer is where data is accessed by applications that visualize and analyze data to get insights and generate reports.

We will store processed data in the serving layer. This includes data in HBase (NoSQL) and Postgres (relational database). The data will be cleaned, processed, aggregated, and summarized.

The data in the serving layer will be used in different ways. Applications like Tableau, PowerBI, and Qlik Sense will use it for reports and insights. Web and mobile apps can access it through APIs for real-time analysis. This layer allows users to choose their preferred tools for analysis and dashboards. Also, tools like HiveQL, SparkSQL, Jupyter, and Apache Airflow will be used for interactive querying, analytics, and reporting. This setup makes data access flexible and efficient for various needs.

8. Conclusion

In this document, we explained how to process data using batch and real-time tools like Apache Nifi, Kafka, Spark Streaming, and Sqoop. We also talked about enabling queries with Hive and Presto, and storing data in HBase and Postgres. The system can grow by adding more nodes, making data processing flexible.

The serving layer makes data easy to access for analysis and reporting. Tools like Tableau, PowerBI, Qlik Sense, HiveQL, SparkSQL, Jupyter, and Apache Airflow help with this. This setup ensures efficient data processing.

Next, we need to set up a project team to manage the implementation and ensure all tools and processes work correctly. This will solve the problems faced by the medical records data processing company.

9. References

- The problem statement is mainly based on the information in the company profile document used for this project
- For the diagram I used lucidchart - https://www.lucidchart.com