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
05/27/2023	Hong Tran	0.1	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
FirstName LastName	Data Architect	Medical Data Processing	student@email.com

1. Purpose

The purpose of this document is to propose a data lake solution that enables newly designed infrastructure of the Medical Data Processing company to be able to scale as the volume of data is significantly growing over the past 3 years.

The document contains:

- Data Lake requirement
- Data Lake architecture design principle
- Assumption
- Data Lake architecture proposal
- Design consideration and rationale
- Conclusion

Target audience of this document:

- Highly technical group of people (enterprise architects, software engineers, technical directors)
- Technical people who are interested in design ideas and decisions at a deep level

In-scope of the project:

- Technical and business requirement
- Design principle and assumption
- Data architecture design
- Rationale behind design decision

Out of scope of the project:

- Implementation of the data architecture replying on Hadoop stack
- Data governance

2. Requirements

❖ **Problem statement:**

The existing infrastructure is applying a monolithic 3-tier application architecture backed by proprietary SQL based databases and warehouses. There is only 1 Master SQL DB Server and 1 Stage SQL DB Server to process the data, and the database currently hosts over 8TB of data coming from multiple sources with various format. So this single node SQL server becomes a ***single point of failure*** causing the entire system down, say when there was a surge in data.

The company can now only scale the database vertically such as creating indexes, upgraded the server hardware and storage configurations. However, this way of ***vertical scaling*** does not make too sense in processing the increasing large amount of data.

Due to the capacity limit of SQL database, data is copied into separate servers on a nightly basis for doing analytics and reporting. This way is posing **challenges in *single version of truth*** and ***data silos.***

- **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 an0d 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
  - Data Volume Growth rate: 15-20% YoY
- **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

➢ These requirements were found in *CompanyProfile-ProblemStatement.docx* document

3. Data Lake Architecture design principles

- **Scalability**: A Data Lake should be designed to be horizontally scalable, and separated out into different layers including Ingestion, Storage, Processing and Serving.

Horizontal scaling helps the server minimize crashing as data volume and velocity increases overtime. And allowing to keep unlimited historical data as presented in technical requirement. It solves the problem that the existing architecture with *vertical scaling* is facing now.

Separated layers allows for modular and scalable architecture, this way we can easily add or remove components as needed, without disrupting the entire system. Therefore, it improves business agility and speed of innovation through automation and ability to experiment with new frameworks as shown in the business requirement.

Because the database currently hosts over 8TB of data, it is recognized as big data problem. Based on the business requirement, the design will leverage open source tools from Apache that allows for distributed processing of large datasets, improving fault tolerance and avoid single point of failure as the existing architecture is currently facing. It also avoids proprietary solutions which can lead to vendor lock-in.

- **Catalog:** a central metadata repository about the actual datasets with descriptions, purpose, schema, and lineage information.

This principle ensures a single version data source of truth and break data silos. Because the current process is extracting data on a nightly basis and maintaining multiple version across departments. This poses a challenge in keeping track of most up to data location over hundreds of tables.

In addition, metadata repository allows to perform Change Data Capture (CDC) process.

- **File format**: use a data format that offers splitability

Processing big data requires breaking the processing jobs into multiple parts. If the files are not splitable, it will reduce processing efficiency since the file can't be broken down into smaller blocks and processed in parallel. Because there are 99% of data sources compressed in gzip format which does not offer split capability, decompose these files for processing is crucial to speed up uptime.

4. Assumptions

- Hadoop cluster will use Linux operating system

  Potential risk:

  - While Hadoop is designed to run on Linux-based operating systems, there can still be compatibility issues with certain components and tools, especially when using older versions of Linux.

- Data Lake will not support data governance because the data lake is designed to be scalable and flexible.

  Potential risk:

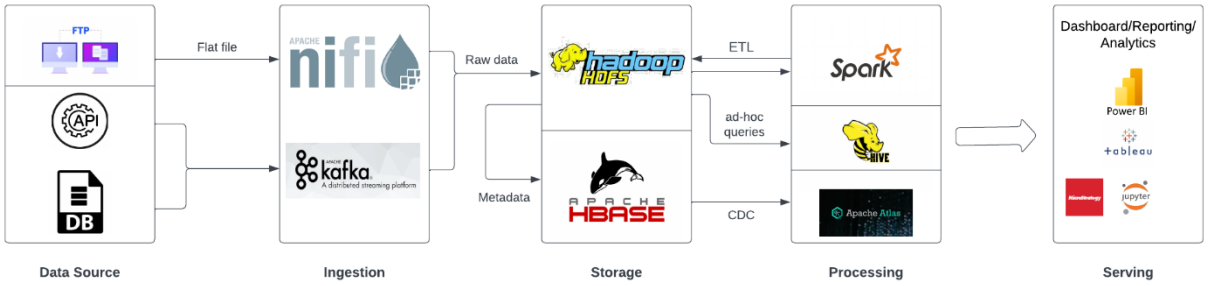
  - Non-compliance with regulations such as GDPR or HIPAA. This can lead to legal and financial penalties for the organization.
  - Inefficient use of resource such as storage and processing power. This can lead to high cost for the company and bad query performance.

- Data coming at a high velocity and need to be ingested into the data lake frequently

  Potential risk:

  - When data is ingested at a high velocity, there is a greater risk of data quality issues such as missing or duplicate data.

5. Data Lake Architecture for Medical Data Processing Company



6. Design Considerations and Rationale

a. Ingestion Layer

How do you plan to ingest different types of data?

- *Batch data ingestion*

  Ingest large data volume

- *Streaming data ingestion*

  Ingest real-time or near real-time data.

How would you ingest data coming from Databases, FTP servers, APIs?

Using Hadoop tools to ingest data coming from Databases, FTP servers, and APIs.

What tools would be used? Why?

❖ *Apache Kafka*

Apache Kafka is selected to ingest data coming from Database and APIs. This tools can handle high-velocity data streams and enable real-time processing. Kafka is horizontally scalable and fault-tolerant.

Considering that the company can only do nightly ETL job with the current architecture, using this tool will help achieve processing data in real-time and enable near real-time dashboard which is CTO's requirement.

❖ *Apache Nifi*

Apache Nifi is selected to ingest data coming from FTP server. It's a data integration tool can be used to ingest data from various sources, including FTP server. Nifi enables to automate the flow of data between different systems. By creating a data flow in Nifi that read files from an FTP server, and then writes them to HDFS. Nifi is highly configurable and can scale out to handle large data volume.

How would the ingestion layer design scale?

As the volume of incoming data increases, we can scale the ingestion layer as follow:

- Kafka

    - Add more Kafka brokers to the cluster to increase its capacity
    - Partition the Kafka topics to distribute the data across multiple Kafka brokers and increase the throughput of the system.
- Nifi
    - Add more Nifi nodes to the cluster to increase its capacity. Make sure to distribute the nodes across different physical machines or virtual instances to ensure high availability and fault tolerance.

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?

Shortcomings of Apache Kafka:

- Not the best choice for batch processing or handling large data volume
- Require a certain level of expertise to set up and configure, which may be a barrier for some users
- No built-in support for data transformation or enrichment, which may require additional tools or custom development.

Shortcomings of Apache Nifi:

- Require more resources to run than other frameworks, which may be a consideration for users with limited hardware resources.
- May not be as widely adopted as other frameworks, which may limit the availability of community support and resources.

AWS services are 3rd party tools could be under consideration as they make up the limitations of Kafka and Nifi for data ingestion as follow:

- **AWS Glue**: provide an easy-to-use visual interface for creating and managing ETL jobs, it also supports batch processing of large data volume.
- **AWS Kinesis**: allow the building of custom applications that process or analyze data in real time.
- **AWS Data Pipeline**: a fully managed service for building and managing data processing workflow. This service also integrates with a wide range of AWS service, so this allows to easily move data between different data stores and processing engines.

Because these AWS services are fully managed that takes care of the underlying infrastructure and maintenance, these 3rd party tools were not selected into the architecture proposal to avoid vendor lock-in.

b. Storage Layer

<How do you plan to store a vast amount of data? >

The company currently hosts over 8TB data with 99% zip files

To store this amount of data in HDFS, we will divide them into smaller chunks and store them as HDFS blocks with each block size in HDFS is 256MB. So we will have 32,768 blocks to store 8TB of data.

Because we can still access the compressed data in HDFS as it were uncompressed, so need to do anything with this matter.

<How would the system handle 20% YoY Data Growth rate?>

It's expected to have 20% of data growth so 1,6TB of data is expected to grow over year.

Two methods to handle with such growth:

- Method 1 (Tiered storage): Save storage space in HDFS

  Store data on different types of storage media based on its access pattern as follow:

  - Store frequently accessed data on SSD
  - Less frequently accessed data on HDD
- Method 2: scale horizontally by allocating more machines to Hadoop cluster

  The amount of data that a machine in Hadoop cluster can process depending on various factors such as hardware configuration, network bandwidth, data block size which is proposed 256MB.

<How do you plan to handle back-up and recovery? What are the strategies?>

Currently the data is backed up to a new database on a nightly basis that usually takes several hours and cause the system offline during the restoring process.

New plan to handle back-up and recovery is using HDFS snapshots to take a point-in-time copy of the data in HDFS.

Strategy:

- Schedule: nightly basis
- Retention period: TBD
  Basically, the retention period is defined based on business requirement (identify which period of time from a point in time is far enough in the past to be useful), and the amount of disk space the company have available (now there is over 3TB available for disk space)
- Test the backups regularly to identify any issues before using that backups in a real disaster.

<How do you plan to store custom metadata information? What type of information would metadata hold?>

Store metadata information using Apache HBase which is a NoSQL database that can store and manage large amounts of structured and semi-structured data.

Metadata would hold data lineage, data ownership, data access control

<What format of the data do you plan to use? Why?>

Data format to use: parquet

Because the system currently hosts over 8TB of data, it could be considered as a big data problem. Big data technology tools such as parquet are optimized to deal with column-oriented format leading to efficient performance. This is because data is stored and retrieved in columns, and hence it can only able to read the relevant data if required. The company currently want to improve uptime of overall system, so this type of storage will help to achieve this requirement.

<How do you plan to secure data (at a high-level)? Identify 2-3 techniques/tools/considerations>

Planning to secure data in Hadoop by authorization (Kerberos) and auditing (HDFS audit logs)

- *Kerberos*: an authorization mechanism in Hadoop that provides secure authentication for client/server applications by using secret-key cryptography. Once the Kerberos infrastructure is set up, we can configure Hadoop to use Kerberos for authentication, which means only authorized users are able to access the data.

  Consideration:

  - Setup Kerberos infrastructure can be complex and time consuming
  - Can have an impact on performance as it adds an extra layer of authentication and encryption to the data access process.
- *HDFS audit logs*: provide a detail record of all file system and user activities. This information can be used to detect and get alert of security incidents such as who is trying to access sensitive data.

  Consideration:

  - Can generate a large amount of data so need to consider storage requirements for these logs.
  - Can have an impact on the performance of Hadoop cluster, as it adds an extra layer of logging to the file system activities.
  - Use appropriate tools to analyze audit logs as it's used for detecting security incident.

<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?>

This tool was considered but not selected and why:

- *Apache Cassandra*: a distributed NoSQL database that can handle large amounts of unstructured data.

  This tool was not selected because Cassandra is not a native Hadoop tool. It is not designed specifically for Hadoop. So it may not be effective and efficient compared to other tools that are designed specifically for Hadoop such as HDFS.

Cassandra is a good choice in the scenarios of high-speed transactional processing and high availability, and it's not optimized for complex analytical query. In contrast, HDFS is optimized to store and manage large amounts of data in a fault-tolerant manner and perform analytical queries on that data.

The business requirement states the system should be fault-tolerant and reliable so HDFS is selected instead of Cassandra.

c. Processing Layer

<How do you plan to process the data?>

Perform various operations on the data including: ETL, interactive query, CDC

<How do you satisfy different processing needs? Batch, Realtime, CDC?>

- Spark can be used for batch and realtime processing
- Atlas can be used to perform CDC by creating a hook that captures changes made to data in HDFS and send that information to Atlas. Atlas then uses this information to update its metadata about the data in HDFS.

<How do you enable ad-hoc querying capabilities?>

Hive can enable ad-hoc querying capabilities as follow:

- Install and configure Hive in Hadoop cluster
- Create external tables in Hive, which map to the data stored in HDFS
- Use HiveQL to perform ad-hoc queries on the data

<What different tools are involved for processing?>

- *Spark*: perform ETL for both batch and streaming processing
- *Hive*: perform ad-hoc query
- *Atlas*: perform CDC

<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?>

Apache Pig and ***Apache Storm*** are two open source tools that were considered to the proposed architecture but they were not selected. This is because ***Pig*** is primarily designed for batch data processing and ***Storm*** is for streaming data processing. However, ***Spark*** bring all of them together, can use spark to do real-time as well as batch data processing use cases under one single unified framework. Therefore, ***Spark*** is selected in the processing layer.

Shortcomings of Spark (tool selected):

- Because Spark relies heavily on in-memory processing, it can lead to memory management issue when there's insufficient available memory to hold the entire dataset and then cause the processing slow down significantly.

Using Pig and Storm together can address the shortcomings of Spark with respect to memory management. Pig does not rely heavily on in-memory processing as it's used for batch processing, so this can help mitigate memory issue when working with large dataset. However, Pig is not designed for low-latency processing, so it's not suitable in the case of the project that requires real-time analytics.

<How does the proposed architecture scale with respect to processing?>

Configure these factors to scale the processing layer:

- Number of nodes in the cluster
- The amount of memory and CPU resources allocated to each node
- The network bandwidth between nodes

For example:

- In Spark, we can use YARN to manage the resources of the cluster
- In Hive, we can do partition and bucket the data stored in HDFS to further optimize query performance.

d. Serving Layer

<What do you mean by serving layer?>

Serving layer serves the data for various applications such as dashboard, visualization, and machine learning. It also provides end-users with access to data stored in the data lake.

<What type of data do you plan to store here?>

- Real-time data that is frequently updated
- Historical data

<How would the data in the serving layer be used?>

Data in the serving layer is used by customers to access data insight in multiple forms, such as near real-time dashboard on top of Tableau and PowerBI, querying to generate custom reports…

8. Conclusion

This document proposes a data lake solution that fully relies on Hadoop tools. The implementation of data lake based on Hadoop is a complex process that may raise challenging to setup and configure the system correctly.

Next steps:

- Investigating the existing technical environment further to have a proper and detailed plan for scaling the system with Hadoop.
- Analyze the current data volume in a deeper manner to calculate data size in each node of the Hadoop cluster, so the system could be scale effectively (now the data size in each node is given 256M).
- Get business requirement to define retention period of data for backup and recovery plan.

9. References

https://www.interviewbit.com/blog/data-lake-architecture/

https://www.techtarget.com/searchdatamanagement/feature/Hadoop-vs-Spark-Comparing-the-two-big-data-frameworks

https://www.knowledgehut.com/blog/big-data/apache-spark-advantages-disadvantages

https://docs.aws.amazon.com/whitepapers/latest/building-data-lakes/data-ingestion-methods.html