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 |
| 09/23/2023 | Rameshkumar Shanmugam | 0.1 | Initial draft |

## Reviewers / Approval

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Version Approved | Position | Date |
|  |  |  |  |

## Key Contacts

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Role | Team | email |
| Rameshkumar Shanmugam | Data Architect | Medical Data Processing | Srameshkumar.shan@gmail.com |

# Note from Instructor:

# Consider this as a comprehensive design document that you will deliver to the technical audience of the company.

# Provide detailed design and implementation level details

# You are expected to provide at least 6 pages worth of content (Does not include the cover (title) page and tracker page)

# Each section has a set of guiding questions that will help you derive the responses.

# Purpose

Medical data processing company’s current architecture is not able to handle the current volume of transactions. As the volume of data continues to grow, the existing single node SQL server is not able to meet company’s business and technical requirements. Currently the company can only process the data nightly due to the compute capacity limitations. ETL processes and SQL reporting queries are running slow due to increased data volumes. Company has optimized the database by creating indexes and upgraded the server hardware to maximum CPU, RAM, and storage configurations that the server can support. However, it has not helped significantly in terms of performance and scale.

The purpose of this document to provide a detailed technical design proposal for an enterprise data lake system to replace existing in-house solution.

* It contains a detailed technical design proposal for an enterprise data lake system. It’s including descriptions of the architecture, explanations of how the proposed design can solve the company’s challenges. And document should clearly state any assumptions or potential risks to the design.
* it is intended to showcase data architecture ability to design a technical solution for an enterprise data lake system and to demonstrate understanding of the business problem and recommendation for a solution.
* it is aimed for technical audience like leadership team, Data engineers and Data scientists

**In scope**: the scope of this document includes business/technical requirements, design principles, assumptions, and data lake architecture.

**Out of scope**: implementation of the data architecture solution

# Requirements

**Summary**:

* + Design a system with high availability and reliability.
  + Easily scalable with increased data volume and velocity.
  + Ad-hoc data analytics, interactive querying capability using SQL.
  + Integrate flexibly with report, dashboards, AI/ML frameworks.

**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: 3,500
* 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
* 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 Power BI, Tableau, or MicroStrategy.
  + Generate daily, weekly, nightly reports using scripts or SQL.
* Ad-hoc data analytics, interactive querying capability using SQL

# Data Lake Architecture design principles

**Simple/Open architecture**: The more complex your data lake architecture is, the more difficult it will be to use. Keep it as simple as possible so that everyone on medical data processing team can easily understand it.

* Store data in open formats like Avro and Parquet which are standard, well-known, and accessible by different tools
* Retain historical data in object storage like Amazon S3. This will allow to cut costs compared to storing data in a database/data warehouse
* Use a central meta-data repository such as AWS Glue or Hive. This will allow to manage meta data in centralized location, which helps to reduce operational costs in infrastructure, IT resources and engineering hours

**Centralized data storage**: Store all ingestion data into one common place so that it will be easy to access and process them for business needs. The type of storage technology you choose will affect the performance of your data lake. Choose the right one for your needs. The most popular options are Amazon Simple Storage Service (S3), Azure Blob Storage, and Google Cloud Storage

**Scalable**: Data Lake solution needs to be able to scale up or down as needed, depending on the amount of data you’re storing and processing. A data lake needs to be scalable so that it can easily handle the increasing volume of data being generated by businesses today. If your data lake isn’t scalable, it will quickly become overwhelmed and unusable

**Performance and reliability:** To ensure high performance when querying data, you need to apply storage best practices to make data widely available

* Every file stored to contain the metadata needed to understand the data structure. Your data lake is being queried by various query engines such as Amazon Athena and Apache Presto. These engines require a schema for querying.
* Use columnar file formats such as Apache Parquet and ORC. Storing your data in columnar format enables you to create metadata for your data, which in turn allows you to understand the structure of your data when querying it. With columnar formats, you can query only the columns you need and avoid scanning redundant data.
* Keep your data in optimal file sizes (compaction). We recommend implementing a Hot/Cold architecture: Hot – small files for good freshness; cold – merging small files into bigger files for better performance.
* Build an efficient partitioning strategy to ensure queries run optimally by only retrieving the relevant data needed to answer a specific analytical question
* Scale horizontally to increase aggregate workload availability
* Automatically recover from failure

**Security**:  One important consideration when designing your data lake is security. You need to make sure that your data is safe and secure, and that only authorized users can access it. There are several ways to secure your data lake, including data encryption, role-based access control and data masking

**Governance**: There are several governance models that can be used for data lakes. The most popular option is the **centralized governance model**. With this model, all the data in the data lake is centrally managed by a single team. This team is responsible for ensuring that the data is properly organized and labeled, and that only authorized users can access it. Data Quality, Data Catalog and Access Control play an important role. Other popular models are distributed governance model and self-governance model

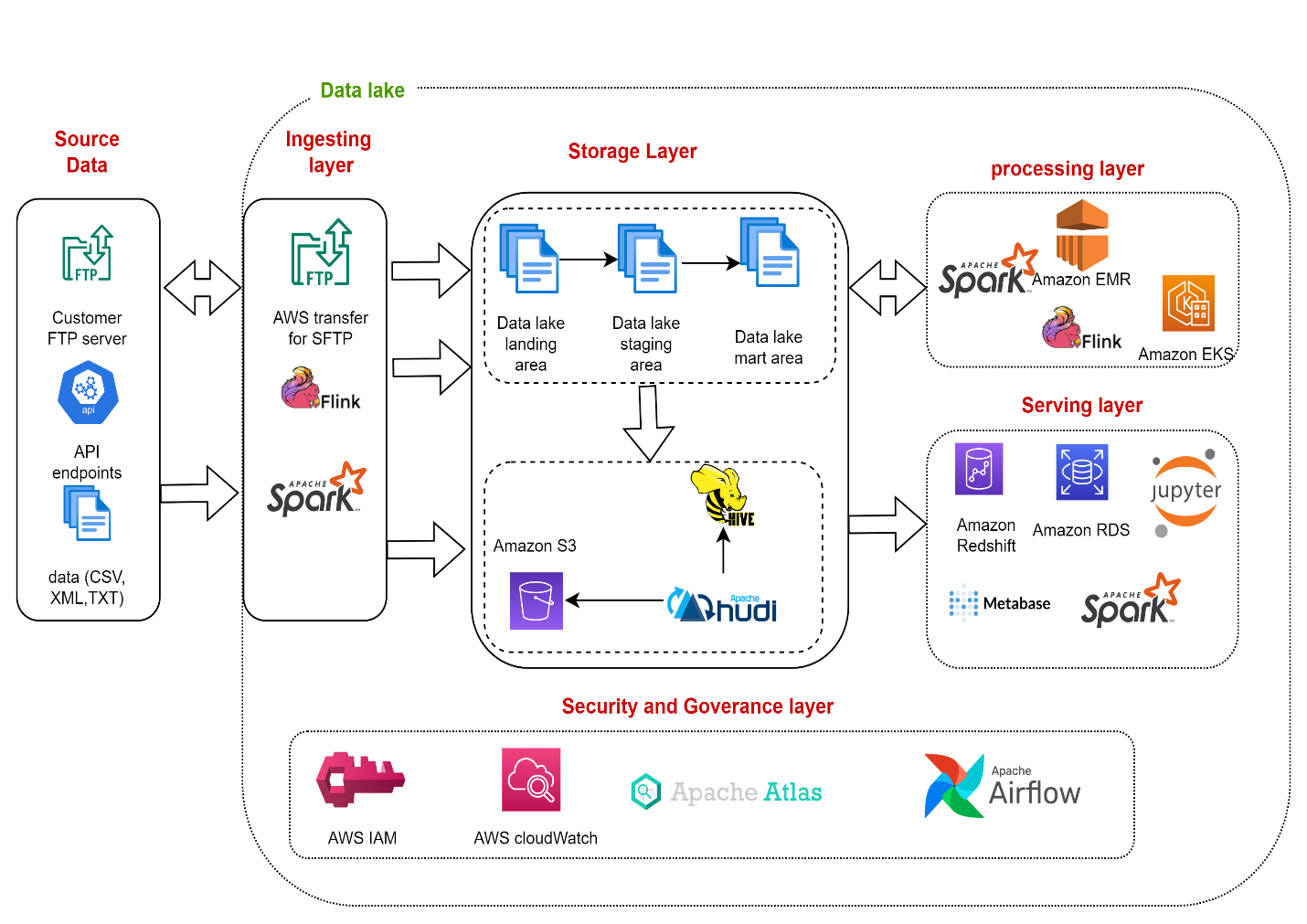
# Assumptions

* Time to switch to data lake as soon as possible
* 100% of data will be moved to the cloud
* Cloud is preferred over on-premises infrastructure. An on-premises data lake imposes challenges. Companies must build their own data pipelines, pay the ongoing management and operational costs in addition to the initial investment on servers and storage equipment, and manually add and configure their servers to scale a data lake to cater to move users or increasing data volume.

# Data Lake Architecture for Medical Data Processing Company

I will leverage the AWS Data lake solution. The overall services can be grouped into the following five categories

* Ingestion layer
* Storage layer
* Processing layer
* Serving layer
* Security and Goverance layer



# Design Considerations and Rationale

## Ingestion Layer

In ingestion layer AWS FTP server for file upload and open source tools Apache spark and Flink will be used.

* **AWS Transfer for SFTP**, a fully managed, highly available SFTP service.it has a fine-grained control over user identity, permissions, and keys. it can help to create users within Transfer for SFTP or make use of an existing identity provider. leverage IAM policies to control the level of access granted to each user. Make use of existing DNS name and SSH public keys, making it easy to migrate to Transfer for SFTP. Existing 8000 individual facilities will continue to connect and to make transfers as usual, with no changes to their existing workflows.
* **Apache Spark** can be used for batch and stream processing of data. You can use it to ingest data into the data lake. Spark can read data from various sources, perform transformations, and write data in Hudi tables.
* **Apache Flink** is an open-source stream processing and batch processing framework for big data processing and analytics. It is designed to process large volumes of data efficiently and reliably in real-time and batch modes.

The following Open sources were considered in this solution but these tools are closely coupled with HDFS file system and Kafka is more like streaming data ingestion tool so these tools are not selected for this data lake solution

* **Sqoop** is a tool designed to transfer data between Hadoop and relational database servers. It is used to import data from relational databases such as MySQL, Oracle to Hadoop HDFS, and export from Hadoop file system to relational databases.
* **Flume** is a tool/service/data ingestion mechanism for collecting aggregating and transporting large amounts of streaming data such as log files, events (etc. . .) from various sources to a centralized data store. Flume is highly reliable, distributed, configurable tool. It is principally designed to copy streaming data (log data) from various web servers to HDFS.
* **Kafka** is designed for distributed high throughput systems, can handle a high volume of data and enables you to pass messages from one end-point to another. Kafka is suitable for both offline and online message consumption. Kafka messages are persisted on the disk and replicated within the cluster to prevent data loss.
* **Nifi** is a realtime data ingestion platform, which can transfer and manage data transfer between sources and destination systems. It supports a wide variety of data formats like logs, geo location data, social feeds, etc.

## Storage Layer

In storage layer, AWS S3 will be primary persistent data store along with open source tools Apache Hive and Hudi used

* This data lake architecture is based on **Amazon S3** as the primary persistent data store. Compute and storage are managed separately and resize on demand with no data loss. It is also highly durable and low-cost with several options to connect. Spreading requests across many connections is a common design pattern to horizontally scale performance. Amazon S3 doesn't have any limits for the number of connections made to the bucket. Data will be stored in the native formats upon ingestion.
* **Apache Hive Metastore** can be used as a metadata repository to store information about datasets, their schemas, and locations.
* **Apache Hudi** is used for incremental data processing, offering record-level insert, update, and delete operations on S3. It helps maintain data quality and offers ACID guarantees.

The Other storage systems HDFS, Azure Blob Storage and Google Cloud Storage are considered and but not selected due to the following reasons

HDFS is open source Hadoop Distributed File System which designed for storing and processing large volumes of data across a cluster of commodity hardware. It is typically used in on-premises Hadoop clusters or in cloud-based Hadoop solutions like Amazon EMR and Azure HDInsight whereas S3 is an object storage service designed for highly scalable and durable storage of data in the cloud. It is not limited to Hadoop or any specific processing framework and can be used for a wide range of storage needs

* HDFS Requires upfront infrastructure investment and ongoing maintenance costs. Costs are relatively fixed and independent of data volume.
* S3 Follows a pay-as-you-go model, where you are charged based on the amount of storage used and data transfer. It can be more cost-effective for variable workloads.

With 20% YoY Data Growth rate, AWS S3 would be cost effective solution when compared to HDFS.

Amazon S3 (Simple Storage Service), Microsoft Azure Blob Storage, and Google Cloud Storage are all popular cloud object storage services offered by leading cloud providers. While they serve a similar fundamental purpose storing and retrieving data. There are some differences in their features, pricing, and ecosystems

## Processing Layer

Amazon EMR or EKS will be used for the most of the data processing activities along with open source tools

* **Apache Spark** For batch processing and complex transformations, Apache Spark can be used. Spark can read and write data directly from Apache Hudi
* **Apache Flink** is an open-source stream processing and batch processing framework for big data processing and analytics. It is designed to process large volumes of data efficiently and reliably in real-time and batch modes
* **Presto** is an open-source, distributed SQL query engine designed for high-performance, interactive querying of large datasets. It was originally developed by Facebook and is now an Apache Software Foundation project. Presto is known for its speed and versatility, making it a popular choice for organizations dealing with big data analytics.
* **Amazon EMR** is a cloud-native big data platform provided by Amazon Web Services (AWS). It simplifies the deployment and management of big data processing frameworks such as Apache Hadoop, Apache Spark, Apache Hive, and more. EMR is designed for processing and analyzing vast amounts of data and is particularly useful for data processing, data transformation, and data analysis tasks. It offers features like automatic scaling, integration with AWS services, and support for popular big data tools, making it an excellent choice for organizations seeking to perform large-scale data processing tasks in a cost-effective and scalable manner.
* **Amazon EKS** is a managed Kubernetes service offered by AWS. Kubernetes is an open-source container orchestration platform used for automating the deployment, scaling, and management of containerized applications. EKS abstracts the complexity of managing the underlying Kubernetes infrastructure, making it easier for organizations to run containerized applications at scale

Apache Hadoop MapReduce, Apache Pig and Apache Hive tools are considered in processing layer but not selected because all of them depends on HDFS to process the data.

* **Apache Hadoop MapReduce** is a batch processing framework that reads data from HDFS, processes it in two phases (Map and Reduce), and writes results back to HDFS. It doesn't provide in-memory processing like Apache spark
* **Pig** is a high-level scripting language for data processing on Hadoop. It translates Pig Latin scripts into MapReduce jobs, providing a more abstract and user-friendly way to process data compared to writing raw MapReduce code
* **Hive** is a data warehouse infrastructure built on top of Hadoop for querying and analyzing large datasets using a SQL-like language called HiveQL

## Serving Layer

End users interact with the data and insights across all the normalized electronic medical data through several ways. For example:

* **Metabase** is a data visualization and business intelligence (BI) tool designed for users who wants to connect to your data lake and allows users to create interactive dashboards, run ad-hoc queries, and generate reports. It provides a user-friendly, non-technical interface for exploring and visualizing data
* **Jupyter Notebook** is an open-source web application that allows users to create and share documents containing live code, equations, visualizations, and narrative text. It's widely used for data analysis, data exploration, and data science tasks
* **Apache Spark** is a powerful distributed data processing framework for scalable data transformations, analytics, and machine learning tasks within the data lake.

## Security and Governance Layer

Governance is key in building a managed data lake. Below are the key services used for governance

* **AWS Identity and Access Management (IAM)**, you can specify who or what can access services and resources in AWS, centrally manage fine-grained permissions, and analyze access to refine permissions across AWS.
* **Apache Atlas** can be used for advanced metadata management and data governance.
* **Apache Airflow** is an open-source platform designed for orchestrating complex workflows and data pipelines. It allows you to schedule, monitor, and manage workflows as directed acyclic graphs (DAGs) of tasks
* **Amazon CloudWatch** collects and visualizes real-time logs, metrics, and event data in automated dashboards to streamline your infrastructure and application maintenance.

# 8. Conclusion

# To summarize, below is the architecture of the proposed data lake:

# A data lake on AWS leverages S3 for secure, cost-effective, durable, and scalable storage. Amazon S3 also offers an extensive set of features to provide strong security for the data lake, including access controls and policies, data transfer over SSL, encryption for data at rest and in motion, logging and monitoring, and more.

# Open source tools Apache Spark, Apache Flink and Presto are used in data ingestion and processing layer

# This architecture provides a scalable, cost-effective, and flexible solution for building a data lake on AWS S3 while leveraging Apache Hudi for incremental data processing and maintaining data quality. It offers flexibility, control, and scalability while using popular open-source tools to achieve the same objectives as other popular vendor(AWS, Azure and GCP) managed Data lake architectures.

# 9. References

* <https://www.montecarlodata.com/blog-data-lake-vs-data-warehouse/>
* <https://dev.to/dmetasoul/4-best-opensource-projects-about-big-data-you-should-try-out-3918>