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 |
| --- | --- | --- | --- |
| 07/05/2023 | Vu Ngo | 0.1 | Initial draft |

## Reviewers / Approval

| Name | Version Approved | Position | Date |
| --- | --- | --- | --- |
| Vu Ngo | 1.0 | Udacity Reviewer  Enterprise Data Lake Architect |  |

## Key Contacts

| Name | Role | Team | email |
| --- | --- | --- | --- |
| Vu Ngo | Data Architect | Medical Data Processing | vu93.ngo@gmail.com |

# 

# Purpose

* Objective: This document is to provide a deep technical understanding of the new Data Lake proposal, for the Medical Data Processing Company.
* The proposal included:
  + Business and Technical requirements.
  + Benefits of datalake.
  + Design of datalake and component of it.
  + Conclusion and reference.
* The current solution in place is not able to meet the Medical Data Processing Company's needs, which is why a new proposition is being made.
* Audience: technical guys (software engineer, devops), BOD (CTO, CEO, CIO)
* Inscope:
  + Ingestion layer
  + Storage layer
  + Processing and serving layer
* Out of scope:
  + Metadata management layer
  + lakehouse layer
  + BI layer (visualization layer).
  + Realtime processing and analysis layer

# Requirements

1. **Company overview:**

* The company specializes in processing various types of Electronic Medical Records (EMR) and **provides real-time insights to medical facilities**.
* Their **data insights help customers** stay compliant with laws, track patient health metrics, and manage bed availability and admission/discharge records.
* The company has **1100 customers, 370 employees, and their solution is used by 8000 individual medical care facilities**.

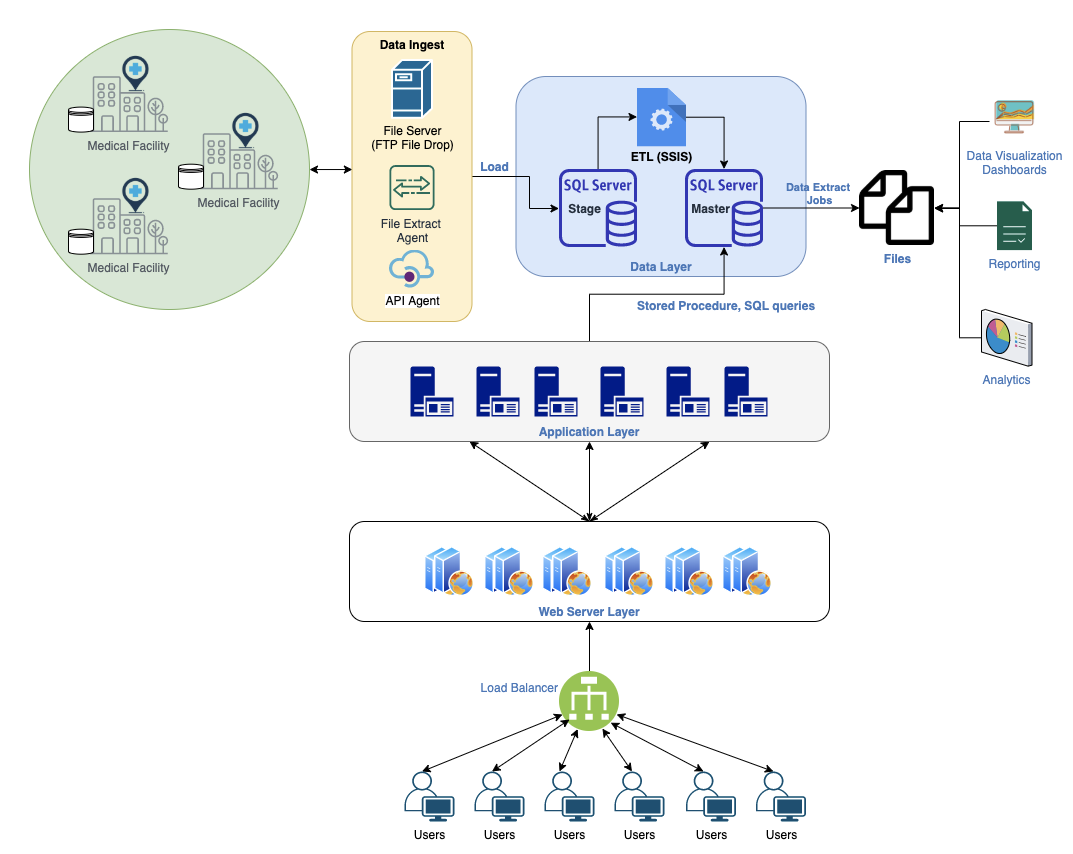
1. **Problem**:

* The company currently hosts over 8TB of data in SQL Server, but the single node server is not able to scale as the volume of data continues to grow.
* ETL processes and SQL reporting queries are running slow due to increased data volumes, and the database server has become a single point of failure.
* Backups are taken on a nightly basis, but restoring them takes hours and systems would be offline during the process, leading to risk and poor customer experience.
* Exporting data on a nightly basis leads to multiple problems such as data duplication, wasted storage space, and data silos within the company, which makes it difficult to build additional capabilities such as Machine Learning models and near-real time dashboards.

1. **Current data architecture**

* Medical Data Processing uses a traditional 3-tier application architecture with proprietary SQL databases and warehouses since its inception in 2008.
* The company processes data in various formats from over 8000 facilities, and the data is retrieved using multiple techniques such as customer APIs and FTP servers.
* Data is transformed, cleaned, deduplicated using proprietary technologies and scripts, and stored in a "master" SQL server database server.
* Customers can access data insights, visualize the data, and generate custom reports using a web-based portal, which is hosted by application servers that query the master database.

Details architecture



# **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

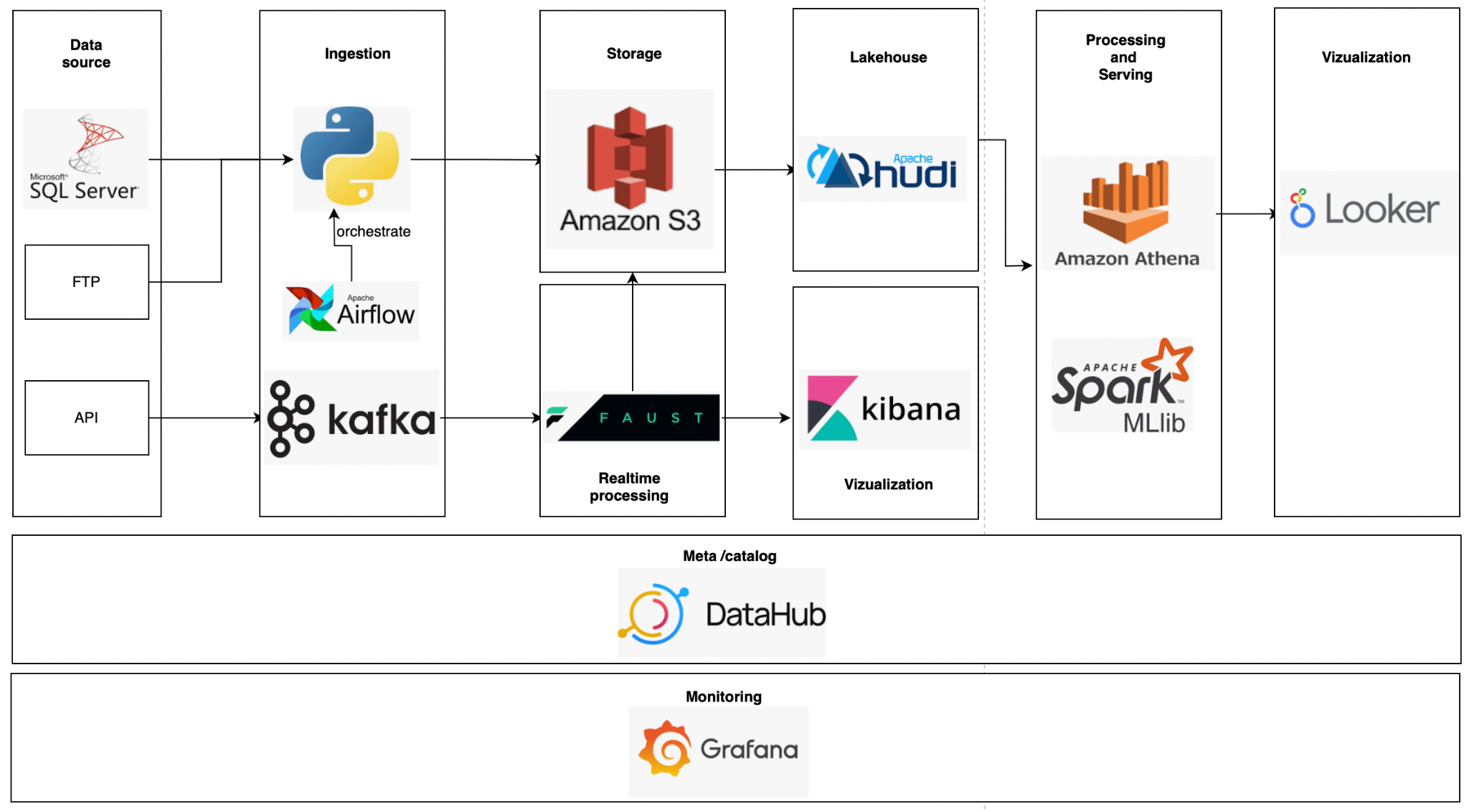
# 3. Data Lake Architecture design principles

* Architecture: The Data Lake should be designed using a scalable, distributed architecture with a separation of the metadata, data, and processing layers. This will allow for efficient processing of incoming files and easy scalability to handle large volumes of data.
* Historical Data: The Data Lake should be able to store and process unlimited historical data, allowing for the retention of all data over time.
* Fault Tolerance: The Data Lake should be designed to sustain a small number of individual node failures without any downtime, ensuring the system is always available.
* Change Data Capture (CDC): The Data Lake should be capable of performing CDC and UPSERT support on a certain number of tables, allowing for real-time updates to the data.
* Multi-Use Case: The Data Lake should be designed to drive multiple use cases from the same dataset, without the need to move or extract the data.
* Integration: The Data Lake should be integrated with different ML frameworks such as TensorFlow, allowing for the creation of machine learning models.
* Reporting: The Data Lake should allow for the creation of dashboards using popular tools such as PowerBI, Tableau, or Microstrategy. It should also support the generation of daily, weekly, and nightly reports using scripts or SQL.
* Analytics: The Data Lake should allow for ad-hoc data analytics and interactive querying capability using SQL, enabling users to perform data analysis quickly and efficiently.
* Security: The Data Lake should design to make sure PII and pathological information has been encrypted

# 4. Assumptions

* The data processing can be divided into smaller, independent tasks that can be executed in parallel.
* Using open source is the good way in this situation when the first time company apply datalake. If the maintenance takes a lot of time we can consider using 3pt later.
* The unlimited historical data storage assumes that the cost of storage is reasonable and feasible for the organization.
* Data quality issues: With the ability to process incoming files on the fly, there may be a risk of introducing low-quality data into the system. It is important to have robust data validation and cleansing processes to mitigate this risk.
* Enough engineer people to implement

# 5. Data Lake Architecture for Medical Data Processing Company



# 6. Design Considerations and Rationale <at least 3 pages>

## Ingestion Layer

* The ingestion layer is a crucial component of a data lake architecture as it serves as the entry point for data into the system. In this specific design, Python and Airflow are utilized to extract data from both FTP and SQL Server sources. Airflow provides the necessary workflow management capabilities to automate the ingestion process, including scheduling and monitoring of data transfers.
* Kafka is also implemented in this design to facilitate real-time data ingestion through APIs. Kafka acts as a high-throughput distributed messaging system that can handle large amounts of real-time data streams. It provides efficient and reliable data delivery, fault-tolerance, and scalability.
* Overall, this ingestion layer design utilizing Python, Airflow, and Kafka provides a robust solution for both batch and real-time data ingestion. The use of these technologies enables efficient data transfer and processing, while ensuring data reliability and scalability, and reducing the potential for errors.
* Some alternative tool:
  + Other workflow management tools such as Apache NiFi could be used instead of Airflow. Additionally.
  + Other streaming platforms such as Apache Flink or Apache Spark Streaming could be used instead of Kafka

## Storage Layer

* The storage layer of the data lake is responsible for storing and managing the data that is ingested from various sources. In this design, the data from FTP and SQL Server is stored on AWS S3, while data from Kafka is processed using Faust and then split into two streams.
* The first stream saves the raw data to S3 using the Boto3 library. This allows for quick and easy access to the raw data, which can be used for a variety of purposes, including backup, recovery, and data analysis.
* The second stream involves real-time processing and visualization using Elasticsearch and Kibana. Elasticsearch provides a highly scalable and distributed search engine that can be used for real-time analytics and search. Kibana is a web-based tool that allows for the visualization of Elasticsearch data through graphs, charts, and dashboards.
* This design uses Faust as a streaming framework to handle the data from Kafka. Faust provides an easy-to-use API for stream processing and allows for the creation of complex event-driven architectures.
* Some alternative tools for the storage layer could include:
  + Apache Hadoop, Apache Spark, or Apache Flink. Hadoop is a distributed data processing framework that can store and process large amounts of data, while Spark and Flink are designed for real-time data processing and stream processing, respectively.
  + Other cloud storage options like Google Cloud Storage or Azure Blob Storage could also be considered. Ultimately, the choice of tool will depend on the specific requirements and constraints of the project.

## Processing Layer

* The processing layer is a crucial part of a data lake architecture as it is where the raw data is transformed into meaningful insights that can be used for analysis and decision-making. In this design, data from S3 will be synced to Apache Hudi, which is an open-source data management framework designed for managing large analytical datasets. Hudi provides the capability to ingest, update and delete data from datasets stored in various file formats in real-time. S3 is a scalable and durable storage system that is suitable for storing large amounts of data, including data in a data lake. However, S3 alone may not be sufficient for some use cases. For example:
  + Data Update and Deletion: Apache Hudi provides the ability to update or delete existing data records, which is not possible in S3. This allows for better data accuracy and integrity, as well as more efficient data processing.
  + Data Indexing: Apache Hudi supports indexing and query acceleration, enabling faster data processing and querying than is possible with S3 alone. This is particularly useful for large-scale data analytics and machine learning use cases.
* Once the data is in Hudi, it can be queried using Amazon Athena which is a serverless, interactive query service that makes it easy to analyze data.
* For machine learning, Spark MLlib can be used which is a scalable machine learning library that includes various algorithms such as classification, regression, clustering, and collaborative filtering. The processed data can then be used to create machine learning models that can be used for predictive analytics.

## Serving Layer

* The serving layer is responsible for serving data to business end users in a way that is efficient, reliable, and scalable. In this design, the data is stored in S3, processed using Apache Hudi, and queried using Amazon Athena. To serve the data to business end users, we will use Looker, a business intelligence and data analytics platform.
* Looker supports direct connectivity to Athena.

# 7. Conclusion

* In conclusion, the designed data lake architecture follows best practices and industry standards, providing a scalable and efficient solution to process, store, and serve large amounts of data. The ingestion layer utilizes Python and Airflow to fetch data from various sources, while Kafka and Faust provide real-time data ingestion capabilities. The storage layer uses AWS S3 and Elasticsearch/Kibana for efficient data storage and real-time visualization. The processing layer leverages Apache Hudi, Amazon Athena, and Spark ML for fast and accurate data processing. Finally, the serving layer utilizes Looker to serve business end-users with intuitive data visualizations.
* Next steps could include testing and validating the designed architecture, identifying any potential bottlenecks or inefficiencies, and continually monitoring and optimizing the system's performance. Additionally, as business needs and requirements change, the architecture may need to be updated or enhanced to support new use cases and data sources.

# 8. References

* Python:<https://www.python.org/>
* Airflow:<https://airflow.apache.org/>
* FTP:<https://en.wikipedia.org/wiki/File_Transfer_Protocol>
* SQL Server:<https://www.microsoft.com/en-us/sql-server/sql-server-2019>
* Kafka:<https://kafka.apache.org/>
* Faust:<https://faust.readthedocs.io/en/latest/>
* AWS S3:<https://aws.amazon.com/s3/>
* Elasticsearch:<https://www.elastic.co/elasticsearch/>
* Kibana:<https://www.elastic.co/kibana>
* Apache Hudi:<https://hudi.apache.org/>
* Amazon Athena:<https://aws.amazon.com/athena/>
* Spark ML:<https://spark.apache.org/mllib/>
* Looker:<https://looker.com/>