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 |
| 01/11/2022 | Daniel Freitas | 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 |
|  |  | Medical Data Processing |  |

# Purpose

The purpose of this document is to provide a deep technical understanding of the new Data Lake proposal for the Medical Data Processing Company. The document contains the architecture decisions that account for the migration from the current process to the Data Lake solution, as well as all the benefits brought by the new proposal.

The current solution in place is not able to meet the Medical Data Processing Company needs and requirements and that is why a new proposition is made.

The target audience are technical people, such as engineers, software architects and related.

In scope of the project:

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

Out of scope of the project:

* Monitoring
* Data quality
* Governance

# Requirements

The requirements were extracted from the Company Profile Problem Statement document and are divided into Business Requirements and Technical Requirements.

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 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 Cognos
  + Generate daily, weekly, nightly reports using scripts or SQL
* Ad-hoc data analytics, interactive querying capability using SQL

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

# Data Lake Architecture design principles

The architecture decisions for the Data Lake solution are based on design principles that must be taken into consideration when developing the entire end to end process.

The Data Lake comes as a solution to accommodate scale of data ingestion and processing of the Medical Data Processing Company. The process in place does not have the ability to do so because it does not use big data tools and frameworks on its backend. Therefore, it won’t scale as needed.

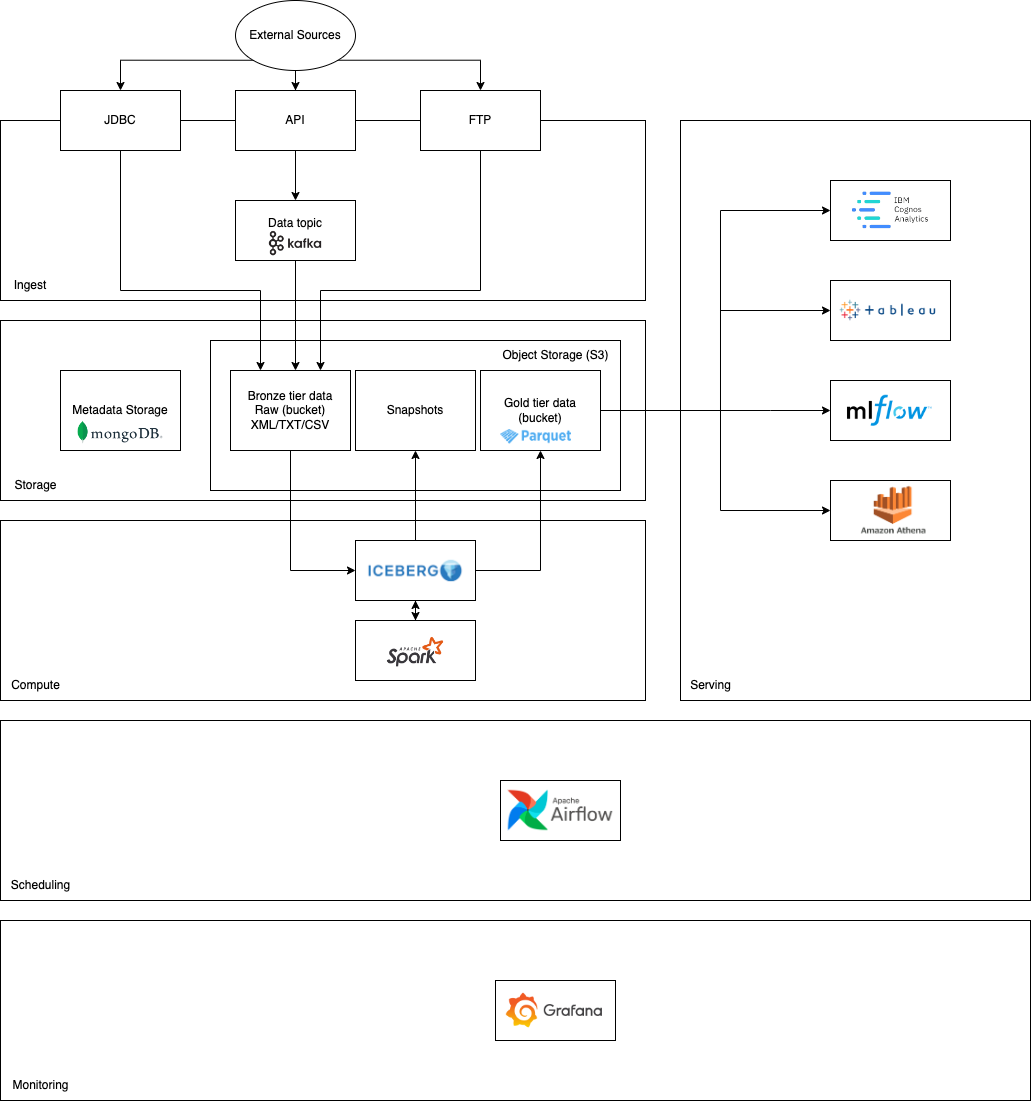
The principles of the new Data Lake solution are:

1. Embrace open-source technologies under the hood to avoid vendor lock-in.
2. Isolate the layers, such as: ingestion, storage, processing, serving, scheduling and monitoring to enable autonomy in each component.
3. Consider using distributed processing where it makes sense to speed up processing time.
4. Consider using streaming processing where it makes sense to speed up analytics.
5. Consider using tools/technologies that scale horizontally to better scale.
6. Consider using monitoring tool everywhere possible to increase observability.
7. Log every part of the process to enable auditing and traceability.
8. Take snapshots to back up the data.
9. Enable routines of data validation to ensure data quality and good data to the end users.

# Assumptions

1. The ingestion of external sources will be made through FTP servers and API. The idea is to use connectors of the Data Lake to ingest the data. Either directly from the FTP server or from the Kafka broker
2. The Airflow orchestrates any task dependency across the entire data flow.
3. Grafana, the observability tool, is plugged at any component across the platform.
4. Snapshots will be taken on the frequency defined in the technical requirements section.
5. Potential risk of the solution is related to data increase. In case of data increase, as the solution horizontally scalable easier than the current platform.
6. Another risk is a little bit of slowness in the user SQL query when compared to the past solution, since it will have to go through the Athena + Iceberg integration.

# Data Lake Architecture for Medical Data Processing Company



# Design Considerations and Rationale

## Ingestion Layer

The interfaces of the ingestion layer should not go through wide changes to allow backward compatibility with the external source patterns.

The platform will be able to ingest XML, CSV and TXT through API, FTP servers and JDBC connector.

In case of streaming data, the API will be the interface of a streaming data. The request is made to the API and the data is published in a Kafka topic to allow real time processing.

The FTP server is also another way to ingest data. The external source push files such as XML, CSV or TXT to the FTP server and the Data Lake pulls it.

When the data comes from relational database, the JDBC connector is the best way to retrieve that data.

Scalability can be achieved by using a proxy with load balancer in the API side. Also, the Kafka cluster can be scaled as needed.

Other open-source tools were considered but were not selected for this project

* **Apache Sqoop**

Apache Sqoop is a tool that integrates RDBMS, used for ingestion and load from databases. Although it could play a role in the ingestion layer in replace of the JDBC connector, the project has been added to the Apache Attic page, what means that the project has reached end of life.

* **Apache Flume**

Apache Flume is an open-source project that leverages a distributed service for moving large amount of log data. It was not selected to be part of the solution as the idea is not to ingest log data from the external sources but other types of files.

* **Apache Nifi**

Apache Nifi is a platform built to automate and manage data flow between systems. Apache Nifi was not considered in the architecture because it has many other components that wouldn’t be useful for our tool, and we decided to go with a lean architecture when it comes to the ingestion part.

## Storage Layer

To achieve a vast amount of data storage we need to leverage a cheap, scalable and reliable storage. The chosen one is an Object Storage like Amazon S3 or IBM Cloud Object Storage that meets the needs listed above. Other than that, the Object Storage works with the type files candidates for ingestion such as CSV, TXT and XML.

There is a 20% of YoY data growth rate and that constraint will be promptly vanished as Object Storage is cheap and simple to scale.

Backup and Recovery

Backup will be produced as snapshots daily, weekly, nightly via Apache Iceberg tool and will be stored in a special bucket in a different cloud region from the production environment, also named as Golden Tier bucket. This way allows data separation.

In case of disaster, the recovery will be made through data transfer from the snapshot to the Golden Tier bucket.

Metadata

Custom metadata information will be maintained in a separate storage given the characteristic of the data. It will be placed in a MongoDB, a NoSQL database. The metadata will hold the data object name, vintage (when it was refreshed), column names and types, size of the table and so on.

Data Format

The data format in the Golden Tier bucket will be Parquet. The reasons behind selecting Parquet are:

* Apache Iceberg supports Parquet format
* Parquet connects well with the tools of the serving layer
* Parquet is efficient on read operations since it is a column based format

Security

The security is assured in the platform due to the following:

* All data is encrypted by design
* Data access depends on the user entitlement

Other tools consideration

We also considered HDFS as a it is a distributed storage solution, but the decision was not to use it because S3 is preferred given how cheap and scalable it is when compared to HDFS.

## Processing Layer

The processing part of the system works as follows:

* + 1. Raw data in the S3 Bronze Tier bucket is ingested by the Apache Iceberg connector which uses Apache Spark for the distributed processing
    2. The process part is made over Apache Spark plan
    3. After the data is transformed, cleaned, deduplicated, Apache Iceberg loads the data back in the storage but now in parquet format in the Golden Tier Bucket

Different Processing needs

* CDC is a capability of Apache Iceberg tables.
* Batch processing will be scheduled and managed by Apache Airflow
* Streaming will go through the same technologies and components, but the dispatch will be handled as soon as the data comes in the S3 Bronze Tier bucket.
* Ad hoc querying capabilities is provided by the Apache Iceberg along with the Golden Tier Bucket. The serving layer hands to the user via Athena which is also integrated with Apache Iceberg.

Other tools consideration

**MapReduce**

MapReduce could be used in replace of the Apache Spark, but MapReduce is considerably slower as it leverages disk IO whereas Apache Spark uses in memory distributed data processing.

**Apache Hudi**

Apache Hudi is an open-source of the modern data stack which also has UPSERT and CDC capabilities. However, we decided to go with Apache Iceberg which is also a widely used tool that deliveries the same features of the Hudi.

## Serving Layer

The serving layer is the part of the system that connects to the storage and provides a self-service and comprehensive tool for end user data consumption.

In the serving layer there is no data stored since every data should be in the Golden Tier Bucket to go through backup and security constraints.

The data in the serving layer will be delivered through connectors to the visualization tools, such as Cognos and Tableau; it will be provided over connectors to the Data Science and AI such as ML Flow; it will be queried by querying engines such as Amazon Athena.

# 8. Conclusion

The system described in this document provides the solution for the existing problem of the Medical Data Processing Company. The current solution does not conform with the technical requirements nor the business requirements of the company. A redesign of the architecture has been provided to accommodate the growth rate of the data as well as the fast data availability that is required by the end users of the platform.

The next steps of the project, in case of approval by the board, initiation of a PoC (Proof of Concept) to put into reality a small version of the project where the users can test the proposed architecture.

# 9. References

Apache Iceberg - <https://github.com/apache/iceberg>

Apache Nifi - <https://nifi.apache.org/>

Apache Spark - <https://spark.apache.org/>

Apache Sqoop - <https://sqoop.apache.org/>

Hadoop - <https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html>

Udacity Data Architecture Course