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 |
| 03/01/2023 | Daniel Pazeto | 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 |

# Purpose

This document has the purpose of provide a new Big Data solution for the company and help the Data Engineer, Data Analysts and Data Science people to achieve their goals with a minimum of failure and downtimes while making reports, data analyzes and ML in a professional way.

It contains all the aspects in a detailed and technical way, including the tools which will be used and how they serve each purpose in order to ingest, transform and serving the data for the generation of reports, APIs and machine learning purpose.

This document does not cover any improvement or change in the databases and FTP servers to be ingested. Any change needed on those external tools will be of responsibility of the service owners.

# Requirements

The current company solution has faced some issue regarding the data quick grow and it have been already reaching almost the maximum of performance and since the company needs this data for several purposes the current solution should be replace for a new one that provide this data.

The current solution is composed of 2 SQL DB Servers, one used for staging data and another used for the final data used for build reports. Also, some smaller server for data ingestions and some web and applications servers.

The data volume ingested growth rate is 15 to 20% Year over year and currently it is 99% zip files within the size between 20KB and 1.5MB, in some case it can have 40MB.

Each file can contain either CSV, TXT and XML records.  
For the XML zip files each zip file can contain 20 to 300 individuals XML files and each XML 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

The main business requirements are listed below

* 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

All those requirements we can find on the **companyprofile-problemstatement-3.docx** document.

# Data Lake Architecture design principles

The Data Lake where is all the data will be storage and consumed from other people and tools, the ingestion could be in batch or (near)real-time and the processing of the data will use the most recent technologies described later. This will be divided in 4 layers which will be Ingestion, Storage, Processing and Serving layer

The storage will contain 3 zones of data:

* Raw zone
* Staging zone
* Trusted zone

The Raw zone will have all the ingested data, here only the developers and administrator will have access and the data will be here as is.

Staging zone will be used as a middle place to save data partially processed and also as a temporary folder for Glue Jobs to save intermediary data before save of the trusted zone.

Trusted zone will contain all the data that will be consumed by other tools and data analysts to generate reports and any other purpose. In this zone the files should be in a right way to be accessible via Athena.

Another important point is that the data will be saved using the hive format path whenever it is possible to improve the processing of any tool that use the data in partitioned way. Below one example of the hive folder format.

* e.g. patient\_status/medical\_unity=1354/patient=8952/year=2022/month=11/day=10/hour=06/

The partitions that will be used should be chosen very carefully to improve the processing of the Spark or queries made on it.

All the orchestration will be made by Apache Airflow.

# Assumptions

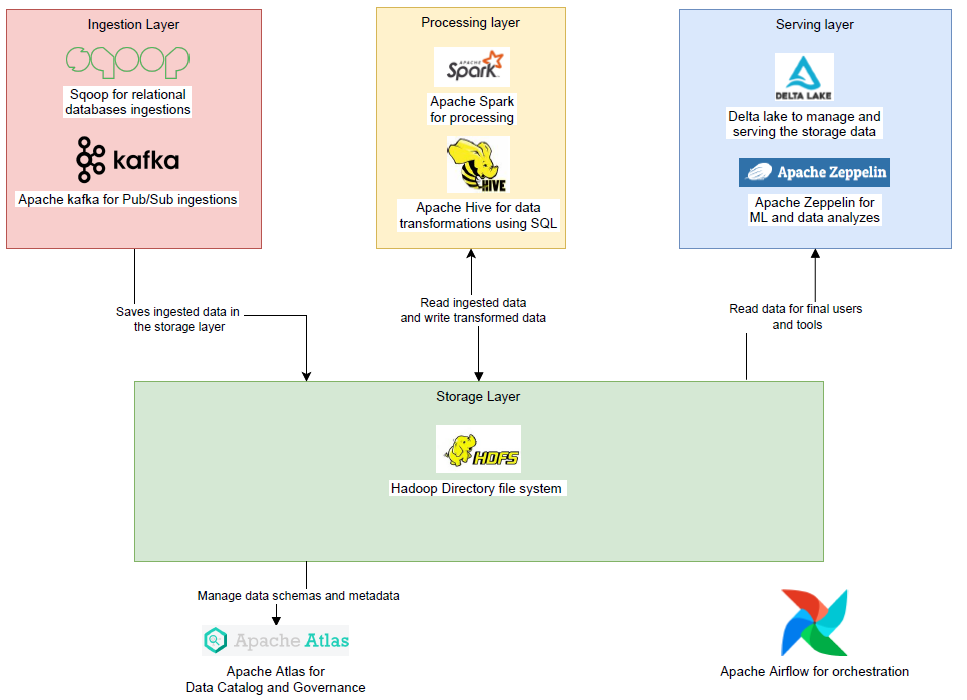
The cloud proposed to be used in this solution will try to use as many open source tools as possible.

This document doesn’t care about costs, the main purpose of it is to solve the business problem. Are the costs of the solution a problem? How much is the monthly/yearly budget for that? Those questions about costs can drive the technologies and also the cloud that will be used for this purpose as well as the tools that regardless its open source it still need some machine to hold those applications and tools.

All the files will be saved in ORC files or if needed compressed JSON (gzip), all of them in folders following the hive folder format path except the metadata. Case this pattern is not followed performance problems and futures or even current refactors will be needed since the most of the bigdata tools nowadays take advantages of partitions and that hive format.

Some minors or not changes will may be needed for the services where this solution will ingest the data, saying that the company and those teams should agreed that the Data Engineering team will not be responsible for those needed changes. Case the company or even the teams aren’t allowed to make some changes in those services this solution can probably fail or found some blocker point.

# Data Lake Architecture for Medical Data Processing Company



# Design Considerations and Rationale

## Ingestion Layer

The data ingestion will depend on the client storage, for relational databases as MySQL and Postgresql there are some solutions to obtain the changes on the tables, ingest it and save on the storage layer, one open source tool for that purpose is the Apache Sqoop[[1]](#endnote-1). The same for NoSQL databases like MongoDB(Mongo change stream[[2]](#endnote-2)). Both solutions will need a machine instance to run and connect to the client database.

For the FTP servers there are also some kind of listeners to let our application knowing when a new file arrived and then they can be ingested and for the APIs we can call them more often passing the date and time of the data we want to ingest.

For all these solutions we will need to use computational instances running any kind of application on it, all the clouds or even on-premise we can obtain these machines, the good thing of using those tools and also the concept of functional data engineering[[3]](#endnote-3), where each instance is responsible for one peace of the data in a specific time is because we can parallelize those extractions and consequently make a horizontally scale up of the resources case needed.

Figure out a scenario where the system should collect 24TB of data in one day, instead the system collects everything in the end of the day this collection is breakdown hourly and them the application now will handle 1TB of data that you can run at the end of the day but every instance of the application running will be responsible for each hour then we can run 24 instances in parallel.   
Also thinking in a scenario where we need this data more often, we can collect every hour the data generated in this last hour and don’t wait anymore to the end of the day.

An alternative tool for the ingestion would be the Amazon Database Migration Service which can get the data from all the SQL databases, but it isn’t cheap and also it can lock the company in the Amazon Cloud.

For the FTP server the Microsoft Cloud has the Data Factory service which allow the ingestion of any FTP server with only few clicks using their web console, but again, these tools have a higher cost and it isn’t open source neither free.

The most famous publisher subscriber tool, Apache Kafka can also be used for all those purposes but the implementation demands some bigger changes on the client side and databases to be feasible, so is better to use the plug and play solutions and avoid as much the needed of changes in the client side.

## Storage Layer

Almost all the data will be storage in HDFS(Hadoop Distributed System), it is a large used storage, can fit the vast amount of data, from gigabytes to petabytes and connects easily with all other mentioned tools in this document, also regardless the expected data growth rate (20% YoY) it can handle well. The own HDFS solution has the replication feature[[4]](#endnote-4) which will help in the backup and recovery needed for the robustness of this solution.

The config files, files with tables schemas, config validations files and all kind of metadata will be saved in this same storage tool. These files don’t need to follow the hive path folder format since the size of them used to be small and don’t need to be separated in partitions. All those metadata can be handle using some tools for this schema versioning and evolving, such as Hudi[[5]](#endnote-5).

In general, all the data saved should be ORC[[6]](#endnote-6)(Optimized Row Columnar) or at least compressed JSON, but the first one is the priority due to high compression rate if compared with other formats as well the better performance for the most of the features which will read and write those files.

Since using the HDFS, this tool has the roles and permissions feature[[7]](#endnote-7) that allow the possibility of deny or allow people to access the data storage on it. For this security and also the data catalog purpose the Atlas[[8]](#endnote-8), which is an open source tool can help.

Some alternatives cloud storage solutions can be used here too such as Amazon S3 and/or GCP Cloud Storage but both of them will have a higher cost and will lock the company on their clouds so the first attempt would be using Apache HDFS that is a open source tools for storage.

## Processing Layer

The data process that will be used are Spark and Hive, those tools can handle the big amount of data and can be triggered by Airflow.

To use them we need the machines instance running at any cloud or even on-premise, which the author does not recommend.

In this layer all data anonymization, cleaning and transformation will be performed, the data will come from the raw folder, will be processed and saved in another folder.

Both the ingested data and processed data should follow the hive folder format and should include the year, month, day and hour as hive folder, also know as partitions. Any other needed partition should be very well evaluated to avoid too many partitions which can diminish the system performance.

Both Spark and Hive has a huge potential to be horizontally scalable due to the parallel processing which would be not a problem in the future regardless the data growth rate.

For this purpose were considered too, Amazon Glue Jobs which runs Spark but this tool have a higher cost and lock the solution to a cloud, in the same way the Azure Data Factory can be used for processing purpose, but for the same reasons the company should first try with the open source solutions.

## Serving Layer

The data can be available in different ways, the more grouped and refined data for reports as well as the rawer data for data analyzes, machine learning, feature engineering and data playground.

A MongoDB can be created to insert the final data for reports and APIs requests, also the Snowflake can be used as a data warehouse and a serving tool for the stakeholders.

Alternatively, using some cloud services the Redshift, Athena on Amazon Cloud and BigQuery on GCP can handle way this layer of give the data to the data analysts and ML people and processing.

# Conclusion

The solution presented in this document will cover the data growth of the business for many years without suffer huge downtimes and the tools provided here are the ones most used in the market making them very connectable with any future requirements and tools.

# References

1. Apache Sqoop. https://sqoop.apache.org/ [↑](#endnote-ref-1)
2. Mongo Change Stream. https://www.mongodb.com/docs/manual/changeStreams/ [↑](#endnote-ref-2)
3. Functional Data Engineering. https://maximebeauchemin.medium.com/functional-data-engineering-a-modern-paradigm-for-batch-data-processing-2327ec32c42a [↑](#endnote-ref-3)
4. HDFS Replication. https://hadoop.apache.org/docs/r1.2.1/hdfs\_design.html#Data+Replication [↑](#endnote-ref-4)
5. Apache Hudi. https://hudi.apache.org/ [↑](#endnote-ref-5)
6. ORC File fomart. https://en.wikipedia.org/wiki/Apache\_ORC [↑](#endnote-ref-6)
7. Hadoop roles and permissions guide. https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsPermissionsGuide.html#Overview [↑](#endnote-ref-7)
8. Atlas. [↑](#endnote-ref-8)