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** |
| **06/12/2021** | Jeegar Maru | 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** |
| **Jeegar Maru** | Data Architect | Medical Data Processing | student@email.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 < approx. ¼ page>

<What is the purpose of the document (Summary)?>

<What does the document contain?>

<Why are you creating this document?>

<Who is the target audience?>

<Inscope and/or out of scope items?>

This document summarizes the requirements & the assumptions & details the design & the underlying decisions for building a Data Lake for the Medical Processing Company to solve the company’s current challenges.

This document is for a technical audience which could be Data Engineers, Data Analysts/Scientists, Architects, etc.

Technical design for all the layers of the Data Lake are in-scope along with requirements summary & any assumptions that have been made.

Discussions/decisions regarding how to accomplish the stated design objectives regarding resources, project timelines, team responsibilities, etc. is out-of-scope of this document.

# Requirements <approx. 1 page>

<Summary of requirements for Data Lake. Summarize your understanding of the problem statement. >

Medical Data Processing company is looking for a scalable, reliable, fault-tolerant system to store Medical-records-related data for easy access, analysis & processing. It should allow the company to automate & innovate freely & not rely on custom scripts for each data-source type by being more metadata-driven. Lastly, it should be able to support multiple use-cases from using MachineLearning to creating dashboards to generating periodic reports using scripts or SQL to allowing interactive adhoc querying using SQL.

<Existing Technical Environment>

The existing technical environment consists of 2 SQL servers of maximum capacity (in terms of CPU & RAM) with a 12TB which is 70% full. There are 70+ ETL jobs running to manage data for 100+ tables. There are 3 smaller servers for FTP server, FTP client agent & API agent to ingest data into the platform from the various medical facilities. Then there is an array of web & application servers to service the customers.

<Current Data Volume>

Medical records data comes from over 8K medical facilities as zip files. 99% of zip files range from 20KB to 1.5MB, but some files can be as large as 40MB. Each zip file contains either XML, CSV or TXT files; for XML, each zip file can contain from 20-300 XML files (where each XML file has 1 record). Average zip files per day is 77K which amounts to 15M data files. This data volume is expected to grow at 15-20% on a yearly basis.

<Business Requirements>

Medical Data Processing company is looking for a reliable, fault-tolerant system to store Medical-records-related data for easy analysis & processing. The system should be scalable as the data volume & velocity increases & should have a better uptime than the current system. It should allow the company to experiment with newer frameworks/tools & to improve business agility & speed of innovation through automation. Also, it should be metadata-driven to avoid building custom scripts for each type of data source.

<Technical Requirements>

The system should be able to process data on the fly (rather than the current nightly batch loads). It should be scalable to store unlimited historical data & to increase processing speed with the increase in data volume. It should be able to sustain small number of nodes going down without any downtime. We should be able to separate the metadata, the medical-records data & the compute & processing layers of the system. It should also be able to capture data changes (CDC) & have UPSERT support for a small number of tables. Lastly, it should be able to support multiple use-cases from using MachineLearning to creating dashboards to generating periodic reports using scripts or SQL to allowing interactive adhoc querying using SQL.

<Where do you find these requirements? Have you seen them somewhere before? >

We typically find these requirements with any data processing company although the data formats & the exact requirements might be different.

# Data Lake Architecture design principles <approx. ½ page>

<List of the design principles. What is the baseline criteria to design the system? What rules/guiding principles should be followed?>

1. Dont move/copy the data too much for each use-case
2. Build a highly available system to minimize downtime
3. Make the design metadata-driven so it’s easy to find the exact data for a use-case

<Provide rationale of “why” the design principal was selected and how would it help in overall Data Lake design long-term for Medical Data Systems>

1. Dont move/copy the data too much for each use-case : Moving or copying the data to different locations results in data duplication & wasted storage space. Moreover, it results in more processing power being utilized & jobs overhead that the team has to maintain, not to mention the development cost to build the said scripts.
2. Build a highly available & fault-tolerant system to minimize downtime : Currently, with just 1 Master SQL server, there is a single point of failure which caused an issue when there was a surge in data last week. A good way to remediate that is to use a distributed system that has some redundancy built into it so that 1 (or a few) node failure does not result in downtime for the system & the applications can continue to function as normal.
3. Make the design metadata-driven so it’s easy to find the exact data for a use-case : Currently, it's difficult, time-consuming & error-prone to find the most up-to-date data location for hundreds of tables & doing that across multiple departments is impossible which creates data silos. A metadata-driven system would allow users to easily find the latest copies of the data for their use-cases across different departments & hence, avoid data silos that harm the company’s ability to analyze & use all it’s data.

[You may not use this example in your final solution]

e.g.: Leverage open source tools as much as possible

# Assumptions <approx. ⅓ page>

<What are the assumptions you have made while creating the Data Lake architecture?>

<Be creative, what questions did you have while designing the architecture?>

<What data is missing in the problem statement, and you made assumptions about it to create the architecture?>

<Describes any potential risks that may be created now or in future based on these assumptions>

[You may not use this example in your final solution] e.g.:

1. Hadoop cluster will use Linux operating system
2. Data Lake will not support X, Y, Z

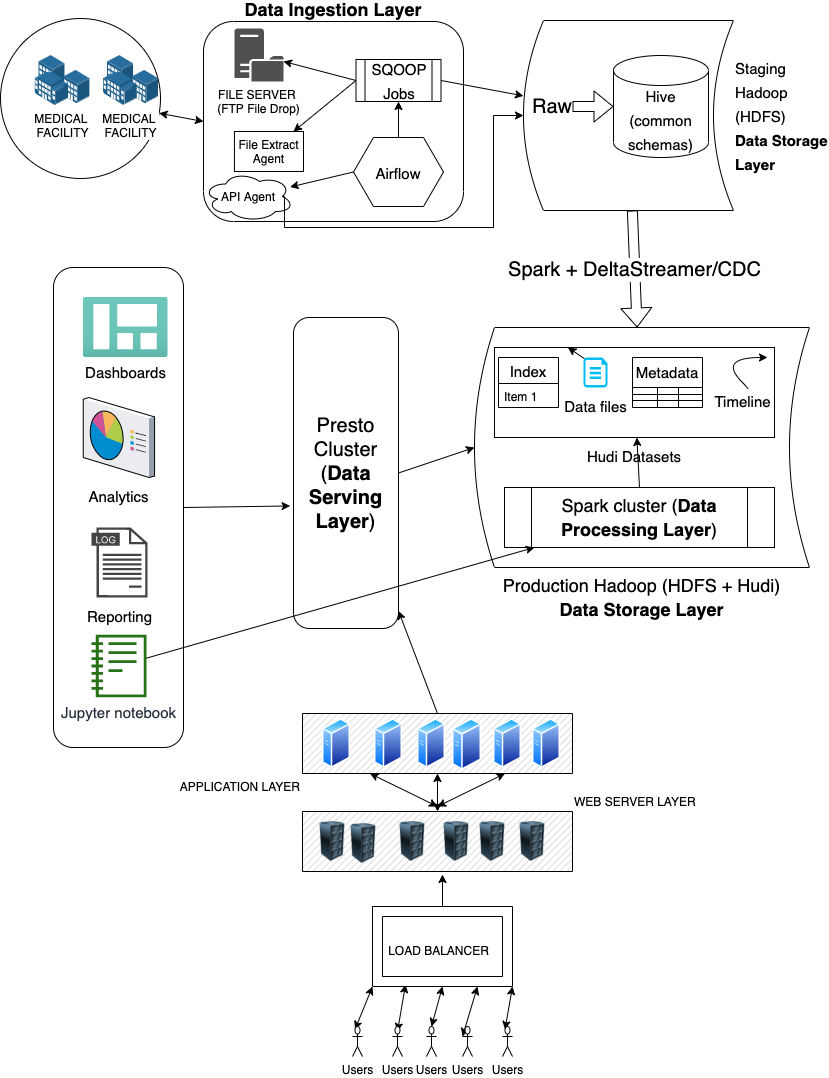
1. Assumption : The company is NOT planning to move to the cloud for this system. Hence, I have not suggested any cloud services/technologies for the problem. Otherwise, I would have suggested AWS, GCS services/technologies if that were the case, which might have provided better features to solve the company’s challenges.

2. Assumption : The XML data structures are not deeply nested & can be normalized to a row & column-based table-like structure. Hence, we can then convert them to a tabular format for storage & analysis purposes.

3. Assumption : We don't need XML-based querying (like using XQuery) for analytics purposes. Otherwise, we would have to consider converting this to JSON, using a NoSQL database (like MongoDB) & supporting analytics on nested data-structures.

# Data Lake Architecture for Medical Data Processing Company

< Embed your Architecture Diagram of Data Lake you created in Step 2 >



# Design Considerations and Rationale <at least 3 pages>

## Ingestion Layer

<How do you plan to ingest different types of data?>

For ingesting different data formats (XML, CSV, TXT), we would define a common standard format/schema that all files will be converted to so that further processing (like deduplication, cleaning, etc.) can be done on the standard format & thereby we don't need many custom scripts to deduplicate, clean & process different data formats.

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

For ingestion, we plan to use Sqoop to ingest data from databases & FTP servers (using Sqoop’s FTP/SFTP connector) & we can use Sqoop Incremental Imports to capture data changes (CDC) for the few tables that require it. For APIs, we could write a small script or a Spark program to ingest that data & store it in our Data Lake.

<What tools would be used? Why? >

Sqoop is an open-source data ingestion tool for Hadoop file-systems (HDFS) that offers fault-tolerance & reliability as it uses Hadoop’s distributed processing & redundant storage to accomplish it’s task so that 1 (or a few) node failures will not affect the job.

We could use Apache Airflow, an open-source, scalable tool to schedule ingestion jobs at near real-time frequency (every few mins) if/when needed. Engineers can define dynamic pipelines or extend the framework with custom operations & it integrates with HDFS.

<How would the ingestion layer design scale?>

As the data volume increases, we could give the Sqoop data ingestion jobs more resources (CPU and/or RAM) on the Hadoop cluster so that it’s able to process the necessary data volume. So, it would be easy to scale the Sqoop job horizontally without requiring system upgrades.

Airflow can scale indefinitely as you can configure the number of workers to handle different jobs & assign them the required resources.

<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?>

I considered using Flume or Kafka for the data ingestion process, but I did not use that as Flume is mainly used for log data which this wasn’t & Kafka is used for real-time streaming data which also we didn’t need for this use-case. I didn’t consider vendor or 3rd party tools as the company prefers to use open-source tools.

## Storage Layer

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

The plan is to store all the data in a Hadoop data lake using the Hadoop Distributed File system (HDFS) which can store an unlimited amount of data via horizontal scaling. This data storage solution is redundant & fault-tolerant as all the data is stored with multiple copies across the distributed file system, so if a few nodes or disks go bad, we will still not lose the data. Also, the solution is cheap as it can be built using commodity hardware. The HDFS storage space will be divided into 2 major spaces - Staging (to hold & clean, deduplicate, process raw data coming from the medical facilities) & Production (to hold the final tables, their snapshots & incremental deltas, meta-data, etc.)

I also plan to use Apache Hudi to manage table snapshots across all the tables to make it easy to get the latest table snapshots or support near-real-time data ingestion or faster loads using Upserts for tables (using the DeltaStreamer) that need change data capture (CDC) as described in the requirements. Moreover, it supports all kinds of use-cases like Snapshot queries (for reports), Incremental queries (for near-real-time use-cases) or Read optimized queries for BI reporting from Tableau, PowerBI, etc. Moreover, Hudi manages file-sizes & layout using statistics which prevents manual work, creates save points for easy data recovery, supports indexing for fast queries, & automatically compacts data files without affecting query performance.

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

To handle the 20% YoY data growth rate, this HDFS data storage layer can grow horizontally by adding new nodes with additional disk storage space. This requires no upgrades on existing nodes & no downtime on the cluster.

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

For backup & restore, we plan to take a nightly backup (or more frequently if the company prefers that) of the latest Hudi snapshots of all the tables stored in the Production Hadoop space. At the same nightly frequency, we would restore this data to a backup Hadoop cluster so that it’s available for use if/when needed in a matter of minutes (rather than hours as it currently takes) if the primary Hadoop cluster goes down/offline. For historical backups, we could move them to a Tape Archive solution to save on costs.

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

Meta-data storage would hold information about all the tables across all departments like the table’s description, column descriptions, how/where to find it's latest snapshot (for analysis), when was it last updated, the type of data source format (XML, csv, txt), it’s lineage, etc. to make it easy to find for everyone to prevent data silos. Also, we could use Hudi’s metadata & timeline features to store some portion of this content, so that Hudi can automate updating it when new data is ingested.

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

In general, we would choose a columnar format (like Parquet) with splittable compression (like Lzo) as Parquet offers efficient querying thanks to predicate pushdown & partitioning with a way to merge schemas (for incremental updates via CDC) which would minimize the amount of data that’s read & processed & hence, save on cost.

Hudi offers either CopyOnWrite or MergeOnRead table-types : We would use CopyOnWrite tables for mostly analytical workload as it produces columnar files (Parquet) that are efficient for analytical queries. We would use MergeOnRead tables for near real-time use-cases without having to copy data to specialized systems & can handle high data volume & data velocity.

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

We would use SFTP to secure the communication between the company & the medical facilities when downloading the data. Internally, we can use encryption at rest when storing the data on HDFS & we can use TLS when reading or writing the data to ensure that data is encrypted across the channel when we process it, query it or use it for analytics. We could also secure sensitive medical data by encrypting PII fields (birth date, SSN, etc.) so that it’s not easy to get at the raw values for bad actors. On HDFS & other tools like Tableau, we would use access control or authentication & authorization to ensure that only the correct users can access the data/dashboard & we’ll ensure that users have the minimum privileges needed for their role.

<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?>

I considered using a Data Warehouse like Snowflake which is great for storing large amounts of structured data & running SQL queries on them. But, it does not handle other analytical use-cases very well like Machine Learning or using Spark for efficient, distributed data processing, etc. I also considered using some AWS products that have some advantages (for eg. S3 supports automatic encryption-at-rest & data processing can be automatically scaled based on the data volume & data velocity using Auto Scaling), but since the company doesn’t seem to be using cloud technologies (or has not expressed an intention to do so), I refrained from suggesting that.

## Processing Layer

<How do you plan to process the data?>

We could use Spark to deduplicate, clean & process the raw data so that it’s ready for Hudi consumption into the Prod HDFS space. Spark is a very efficient, distributed data processing framework that can scale automatically based on data volume & supports many different use-cases including Machine Learning. Also, users can use Spark SQL or Spark ML on the Production Hadoop cluster (with Hudi datasets) to perform adhoc analysis or build models, etc. We could use Apache Airflow to handle scheduling ETL jobs to ingest the raw data & process them (using Spark) to make them available in the Data Lake for consumption by other services.

For custom analytics, we could use Jupyter notebooks with Scala or Python to process & analyze the data. We could use any number of Machine learning or data science libraries/frameworks like TensorFlow, scikit-learn, Spark ML, etc. to build ML models.

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

I think Spark is a great tool for Batch processing across large datasets (in TBs as is in this case) to create reports or to deduplicate & clean the data, etc. For real-time needs or for CDC needs, we could use Spark Streaming or Apache Flink which could handle incremental data with high velocity & provide real-time dashboard & analytics.

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

I would recommend using Presto for adhoc SQL-based querying which is a distributed open-source SQL query engine (with performance comparable to commercial data warehouses) & can handle datasets in GBs to PBs. It integrates with data stored in Hadoop HDFS directly & can combine data from multiple sources.

<What different tools are involved for processing?>

That’s Spark, Spark Streaming or Apache Flink, Presto & Jupyter notebooks (for adhoc analysis) & other ML libraries/frameworks like Tensorflow for developing ML models.

<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?>

I considered using Hive & Pig for data processing but these tools are slower than Spark (since they write the data to disk after every step) & are mainly based in Java & hence don’t provide the flexibility of using Python/R & other libraries/frameworks (like scikit-learn, Jupyter).

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

Spark or Spark Streaming support dynamic allocation where the processing automatically scales based on the data processing volume & can be scaled manually if required. Spark streaming can process data at millisecond latencies which would be great in case there is high velocity data. Flink & Presto are also distributed tools that can handle large data volumes.

## Serving Layer

<What do you mean by serving layer?>

Serving layer would store the final, clean, deduplicated data across all the tables & departments that customers/users can query from for their various use-cases (like near-real-time or adhoc querying, etc.). There would be no inaccurate/incomplete data in the serving layer.

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

You ideally want all the tables across all the departments represented in the serving layer so that the most frequent use-cases (be it for reporting, near real-time, etc.) are efficiently addressed. This could mean denormalized tables or fast HUDI-based indexing or creating data marts to address specific use-cases.

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

The data in the serving layer would be used by end-users or applications. We plan to use a Presto cluster to provide efficient distributed SQL access for reporting & adhoc SQL querying needs (including the web application accessed by Users). Data Analyst or BI tools like Tableau or PowerBI would access the clean data in the serving layer via the Presto cluster too to build dashboards or real-time reports for the C-level executives. Data Analysts & data Scientists would use the data in the serving layer to reconcile data differences between different departments & run accurate analysis or build accurate models.

# 8. Conclusion <approx 2-5 lines>

<Conclude the contents of the document. Provide recommendations on next steps if any.>

In order to solve the company’s challenges when it comes to managing data effectively, we could use a Hadoop-based Data Lake with Apache Hudi that would provide a reliable, fault-tolerant & scalable data solution for the company’s needs. With open-source technologies like Sqoop for data ingestion, Airflow for job scheduling, Spark for data processing & Presto for SQL queries, we can process incoming files on the fly & support special use-cases like CDC & UPSERT & improve business agility through automation & support various use-cases from near-real-time processing to reporting dashboards via Tableau/PowerBI to Machine Learning.

# 9. References <If any>

<Provide links of any external documentation, wiki, blogs that you used to complete your research to put this solution together>

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

Sqoop Incremental Imports : <https://sqoop.apache.org/docs/1.4.2/SqoopUserGuide.html#_incremental_imports>

Sqoop FTP connector : <https://sqoop.apache.org/docs/1.99.7/user/connectors/Connector-FTP.html>

Airflow : <https://airflow.apache.org/>

Hudi homepage : <https://hudi.apache.org/>

Hudi use-cases : <https://hudi.apache.org/docs/use_cases.html>

Hudi concepts : <https://hudi.apache.org/docs/concepts.html>

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

Presto : <https://prestodb.io/>

Parquet : <https://parquet.apache.org/>

Jupyter : <https://jupyter.org/>