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 |
| 09/07/2023 | Uyen Do Thi Kim | 0.1 | Initial draft |
| 09/09/2023 | Uyen Do Thi Kim | 0.2 | Add next steps |

## Reviewers / Approval

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Version Approved | Position | Date |
| FirstName LastName | 1.0 | Udacity Reviewer  Enterprise Data Lake Architect |  |

## Key Contacts

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Role | Team | email |
| Uyen Do Thi Kim | Data Architect | Medical Data Processing | Uyendtk2@fpt.com |

# Purpose

The purpose of the architect document is to design the data lake architecture for the Medical Data Processing Company (MDPC).

The company is currently facing a lot of issues while storing and processing data with the existing data architect, that the CTO of MDPC looked for an alternative architecture. He asked if we could build a data lake to solve current data problems of MPDC, and also to do real-time processing to discover more insights or apply Machine Learning.

This document contains the following details:

* The technical and business requirements of the data lake design
* Principles of data lake design
* Architecture diagram to clarify the idea of the data lake
* Assumptions when designing the data lake.
* Consideration and rationale
* Conclusion

The document would be reported to CTO, and of course all technical data professionals of MPDC.

Items Out of Scope:

* + Do not include code/ scripts for each of the service used in the designed architecture. Small demo is also not in scope.
  + Data governance also is not in scope for this design.

# Requirements

In all, the CTO of MPDC want a scaled data architecture that could be adapted to the increase of data, and could reduce the low performance, risk and wasted storage, that those issues are facing in their current architecture on premises.

We can summarize the requirements as follows:

• Real-time ingestion and processing of incoming data

• Fault tolerance at high level

• Processing scalability at high level

• Ability to store all historical data without size limits

• An UI to interact with data after processed through the SQL syntax

\*\*Existing Technical Environment

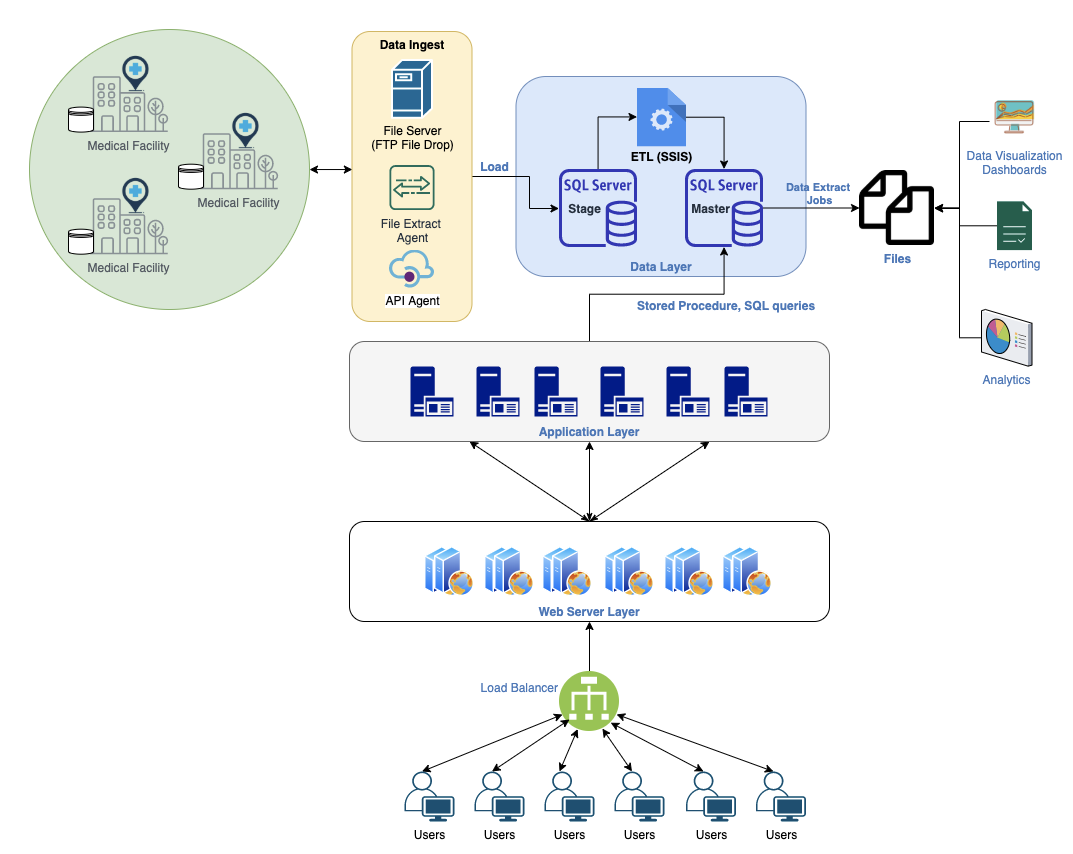
The existing technical environments contains 4 big parts: The Master SQL DB server, the Stage SQL DB Server, the 3 smaller servers for Data Ingestion and the series of web and application servers.

The Stage SQL DB Server is currently run with 64 core vCPU, 512 GB RAM, 12 TB disk space (70% full, ~8.4TB). And this Stage server is now support over 100 tables with 70+ ETL job running.

The 3 servers for Data Ingestion are the FTP server, data and API extract agents.

Those servers of web and application is now run with 32GB Ram each server with 16 core vCPUs.

Below diagram to show the architect of MPDC current data system on premises.



\*\*Current Data Volume

Current data volume came from over 8000 facilities, which is mostly archived into zip files with size ranges from 20KB to 1.5MB and just several cases is archived into zip files with size around 40MB. Those records data is saved in CSV, TXT, XML format before archived as zip files. The case of XML zip files, each zip files contains around 20-300 XML files inside, while each XML file contains only one record.

The daily volume data that data system have to process is average 77 thousand zip files and 15 million data files. The volume hourly is average 3.5 thousand zip files and 700 thousand data files. And every year the data volume gradually increases 15-20 percentage.

\*\*Business Requirements

The business requirements can be summarized as follows:

* + Continuously processing data on demand.
  + Reduced latency of SQL queries
  + Reliability and fault tolerance
  + The system should scale with bigger and faster data
  + Automation & capability to use new frameworks
  + Usage of open-source tools whenever possible to avoid vendor lock-in
  + Centralization of the enterprise data assets with ease of access

\*\*Technical Requirements

The technical requirements can be summarized as follows:

* + The system can process incoming files on demand (not only night batch).
  + Create metadata, create separate layers for data processing.
  + Historical data of all year would be kept with not limit.
  + Be able to scale up the system up to speed up performance when data volume is increased.
  + No downtime when the system just processes small amount of data
  + Be able to implement Change data Capture (CDC) and UPSERT for some certain tables.
  + Be able to do some use cases for advanced analytics, such as integrate ML framework such as TensorFlow, or could integrate with visualization tool to create dashboard such as Power BI, Tableau, also could generate scripts or SQL for daily, weekly, nightly reports.
  + Be able to do some ad-hoc tasks for analytics, interactive SQL querying capacity.

All above requirements are mentioned in the company profile and in the problem statement provided in the project’s resources.

# Data Lake Architecture design principles

To design the data lake, we rely on the following principles:

* Centralized storage and processing: All incoming enterprise data (structured, semi-structured, and unstructured) are stored and processed in a single central location. This is aligned with one of the key business requirements collected from MDPC’s leadership.
* Unlimited scale: The data lake should have the capability to store data at an unlimited scale. It means the data system is able to handle and process a large amount of data from many sources at them same time. This should be aligned with unlimited storage of historical data and architecture should scale as data volume and data velocity increase. While providing data on time, the system must ensure the data accuracy and consistent.
* The new data system would adapt to provide new insights and analytics services: the system should be strong enough to provide data continuously for real-time data monitoring, analytics dashboard, and also provide latest data to feed the Machine Learning model.
* The necessary use of open-source tool: currently the use of open-source tool from Apache system is increasingly and their communities have strongly grown recently. Besides, it brings low-cost benefit, with the ability the same as cloud service. This will have the administrator can easily control the system and change configuration if any.

# Assumptions

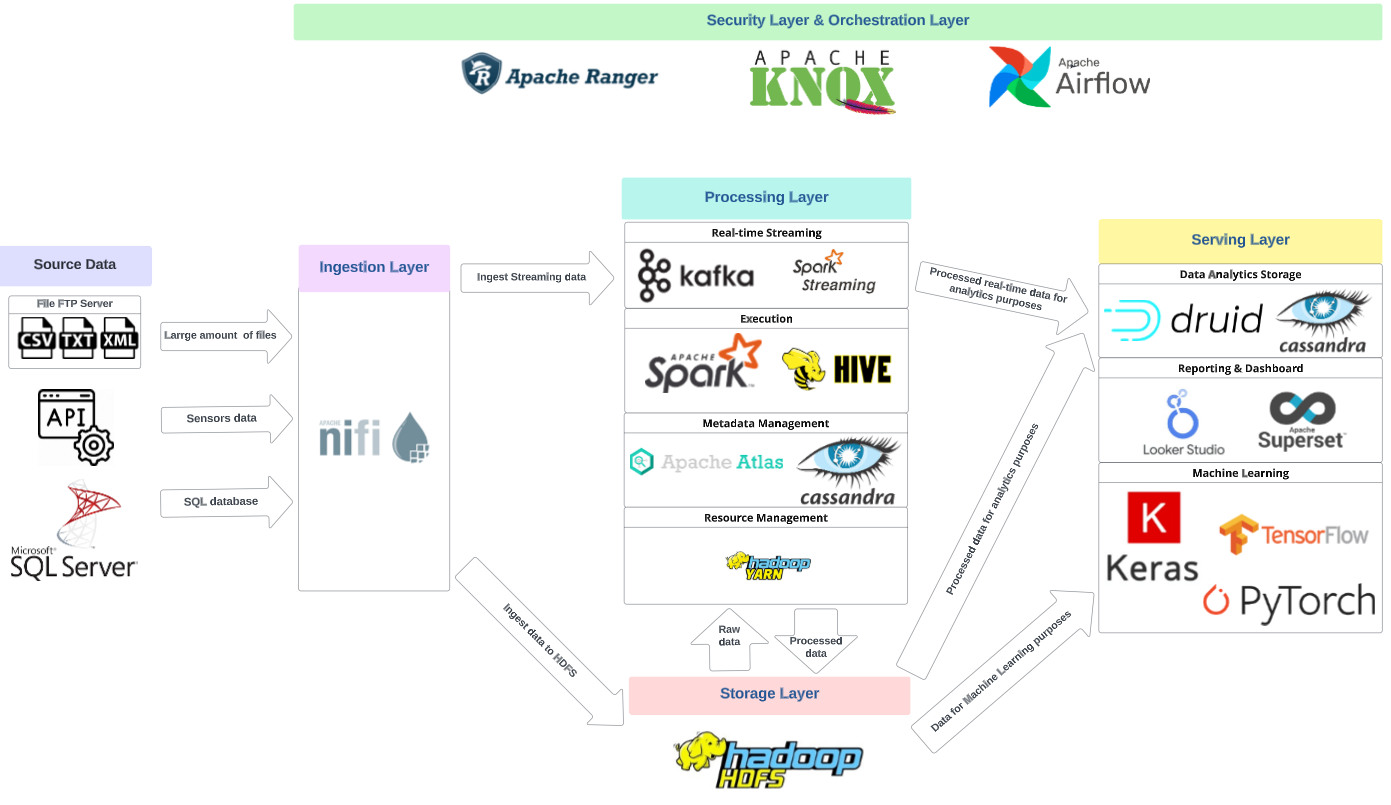
Assume that we would use the open-source tools from Apache system, MPDC has the professionals with strong engineering skills that can build, configure, handle, and monitor the data system. They can also handle the data migration from the old system to the new one with no downtime.

More than that, in the problem document, I could not see any to mention about the data quality management. So, data governance is not a must to be present in the architect solution.

As support for analytics purposes, the data storage would adapt to connect with visualization tools such as Tableau, Superset, Looker Studio to allowed customers to create interactive reports.

While designing the architecture, I have to consider about the increase of operational risk in order to meet a higher number of complex business requirements. Moreover, I have to think about the complexity of managing the user permission from different business unit or other clients if they request to access. I also think about the operational log while running processing data for this data lake, and also what kind of log should be saved. Although data security also is not in scope, but it is important to think about.

# Data Lake Architecture for Medical Data Processing Company



# Design Considerations and Rationale

## Ingestion Layer

Apache Nifi is used for the ingestion layer as it has many advantages.

* Nifi provides a user-friendly web-based UI, which provides seamless experience for developers, or customers to design, control, and monitor the ingestion tasks intuitively.
* Apache Nifi supports a wide array of data formats such as XML, TXT, CSV of logs and geo location data from sensors. Nifi can also handle anything that can be accessed via an HTTPS or APIs. Apache Nifi supports several different protocols including HTTP/S, SFTP, HDFS, and several different messaging systems, such as Apache Kafka, and most major databases. This means that FTP server, API data from EMR sensor, or even SQL server database are supported, making this platform popular for professionals to deal with massive data lakes and complex data flows through different processes.
* Apache Nifi can handle large volume of data with high velocity based on its master-slave node architecture through storing data across nodes.
* Base on flexible scaling model, Nifi is designed to scale-out through the use of clustering on many nodes together. Nifi is also designed to scale-up and down flexibly. So, this means when scale up, it allows more processes to execute simultaneously, providing greater throughput, and when scale down, it allows to run with low throughput.
* Apache Nifi also supports real-time data streaming. When the data stream has been processed, you can relocate data to many destinations by utilizing Nifi’s custom processor. Obviously, data streaming requirement from our system could be facilitated by Nifi.
* Apache Nifi supports data compression at the output, making data storage more efficient than uncompressed data storage.
* Nifi provides processors specifically designed for Change Data Capture (CDC) that allows to track changes in databases and process change data.

## Storage Layer

This layer is the most important to the whole data system. As HDFS acts as a storage layer for the whole Hadoop stack, so it would be the most suitable service to use as a storage layer.

* HDFS is designed with ability to scale to handle big data on a distributed cluster. When data grow larger, we can add more nodes to cluster. So, with 20% YoY Data Growth rate, HDFS can easily handle.
* HDFS can replicate data across nodes. When the data node fails, applications are redirected to other nodes which have the data. The master node also initiates replication of data on a live node to maintain replication levels. This fault-tolerance mechanism makes the HDFS handle back-up and recovery.
* The HDFS architecture allows the master node to hold the metadata about the data stored on data nodes, does not contain actual data.
* To secure data, HDFS has the transparent encryption end-to-end. That means the data to be read from and written to the special HDFS directories is encrypted and decrypted transparently without requiring any changes to the user application code. Also, the Key Management Server (KMS) mechanism of Hadoop system could manage user’s permissions to access data after authorization.
* To satisfy possible future customer needs, our storage layer should have the capability to store data formats like PNG, JPEG (x-ray scans), JSON, Avro, parquet, ORC, CSV, TXT and XML. While the HDFS supports various formats included those listed formats.

## Processing Layer

To handle the streaming data for specific sync up tasks between systems, Apache Kafka and Apache Spark Streaming are used.

* Kafka is a scalable, high performance, low latency platform that can handle real-time data ingestion and processing like messaging system. In data architecture for MPDC, Kafka would be the tool to handle the real-time data that needs simple or very less processing steps. Apache Kafka also supports Change Data Capture, means that it can be used to capture and stream data changes in real-time.
* Spark Streaming is part of the Apache Spark platform that enables scalable, high throughput, fault tolerant processing real-time data. Spark Streaming divides data into small batches and processes them using Spark's parallel processing engine. In MPDC’s system, Spark Streaming would handle the real-time data that needs complex processing steps for analytics or machine learning purposes.

For execute the processing/ETL tasks, Apache Spark and Apache Hive are used.

* Apache Spark is an in-memory distributed computing engine designed for fast data processing. Spark provides built-in fault tolerance and has ability to scale horizontally by distributing the processing data across a cluster of machines. In case of MPDC’s data system, Spark is best suited not only for batch processing but also for real-time data processing (with Spark Streaming) which ensures fast and accurate data computation on large amounts of data.
* Apache Hive is a data warehouse service build on top of Hadoop to handle large datasets with high scalability. Hive supports both structured and semi-structured data. It also provides schema-on-read technique. Hive provides HiveSQL query language to query and analyze large datasets and obviously can implement ad-hoc queries for processing. Hive also supports data partitioning and bucketing, as a result it can improve query performance significantly. In MPDC data system, Hive would be suitable for batch processing for large-scale data analysis and data warehouse.

Spark and Hive would be a good combination to improve capacity to handle big data for data warehouse and analysis. While Spark is ideally fit for real-time processing, Hive plays an important role to process and import historical data for data warehousing.

Although Spark and Hive do not have built-in support for Change Data Capture (CDC) in execution layer, we can configure Spark to implement CDC by leveraging Kafka. Kafka can act as a central hub to capture change events from various sources, then Spark can consume these change events from Kafka topics and process them accordingly.

To manage the metadata, the Apache Atlas and Apache Cassandra are chosen.

* Apache Atlas could support the governance services for our system. But the Atlas is mainly for the Metadata management. Atlas helps capture metadata object details and their relationships. Moreover, Atlas helps with the security of metadata access by enabling controls on data access. Besides, Atlas could integrate with Apache Ranger to enables authorization/data-masking for metadata. In case of MDPC system, Atlas would hold metadata from Spark (including Spark Streaming) and Hive.
* Apache Cassandra is a highly scalable and distributed NoSQL database that support Atlas in handling the large amount of metadata storage. While Cassandra supports the unstructured data, that means it allows flexible schema design, which is suitable to deal with changing metadata structures.

Finally, the Apache YARN is used to handle the resource management tasks. Apache YARN is primarily designed for resource management like CPU, RAM, etc. for Spark and Hive, making the resources allocated simultaneously and jobs coordinated between Hadoop services. Kafka, Atlas and Cassandra can be integrated with YARN for improving their resource management.

## Serving Layer

To store the analytics data for specific analytics purposes, the Apache Druid and Apache Cassandra are used.

* Apache Druid is a high performance, column-oriented database that allows to build real-time analytics data warehouse. It is designed for fast querying and aggregations on large datasets. Druid also provides the ability to query large amount of data by executing OLAP queries to get billions to trillions of rows without pre-defining or caching queries in advance (schema-on-read). In all, Druid is the most suitable as an analytical database, because it can ingest the historical processed data from HDFS through batch ingestion and can hold real-time processed data from Streaming Layer through real-time ingestion. In MPDC’s case, Druid is most suitable as a data warehouse for real-time and historical data by benefiting its fast- querying capability.
* Apache Cassandra plays a crucial role in this layer, because it can act as an archival storage solution for historical less-accessed data in Druid. Druid is optimized for fast querying on recent data, but not a good solution to store and query large amount of historical less-accessed data. By archiving historical less-accessed data in Cassandra, we can free up resources in Druid to keep recent and active data, and still get access to less-accessed data whenever needed.

For visualization purposes, free service Looker Studio or open-source tool Apache Superset can be used.

* Looker Studio is a free visualization service from Google. It can access to a wide variety of data sources. It also helps the analytics professionals to build beautiful dashboards and share reports with stakeholders.
* Apache Superset is an open-source tool from Apache organization for visualization. It provides an interface with attractive charts and can perform custom SQL queries for specific analytics purposes. While this tool can connect with various sources, it also supports the real-time data reports.

To adapt with the purpose to get more insights from data, some services or frameworks or languages can be considered such as Pytorch, Tensorflow, or Keras in other to run some Machine Learning algorithms. This would help the Data Scientists implement some advanced analytics such as forecasting and clustering.

## Security Layer and Orchestration Layer

In this layer, I use Apache Ranger and Apache Knox for security, and Apache Airflow for Orchestration for the whole data system.

The Apache Ranger would help manage user control and authorization for the whole data system. Ranger allows to centralize security administration for the Hadoop cluster. Ranger can support different authorization methods such as role-based access control and attribute-based access control. Besides, it helps centralize auditing of user access and administrative security-related actions, within all the Hadoop components.

Apache Knox would be much helpful with its authentication feature for our system. The Knox Gateway provides a single access point for all REST and HTTP interactions with Apache Hadoop clusters. We could make use of the Proxying service, Authentication services such as authentication for Rest APIs access, WebSSO flow for all UIs, Knox Token, and Client SDK.

Apache Airflow provides an interface that makes it easy to visualize and manage the entire workflow by setting up many Direct Acyclic Graph (DAGs), ensuring that tasks are executed in the correct order and with the necessary dependencies. Airflow is designed to handle large-scale workflows and can scale to accommodate increased workloads then improve overall workflow performance. Airflow can be integrated with various data sources, database, data platform through a rich set of operators and hooks.

# 8. Conclusion

In this document, I presented a scalable data lake architecture that accommodates with MDPC’s growing data amount.

The data lake consists of 4 main layers: ingestion, storage, processing, and serving. And the security and orchestration layer could be used whenever needed. The storage layer in the data lake uses HDFS which allows scale data storage automatically. The processing layer consists of a few efficient, strong and suitable tools at no cost of Hadoop system such as Kafka, Spark, Hive, Atlas, Cassandra, and YARN. Finally, the serving layer democratizes analytics data access through dashboards (Looker Studio/ Superset), while stores analytics data in Druid/ Cassandra database and adapts with Machine Learning purpose through using Keras, Tensorflow or Pytorch framework.

The next step is that the CTO should let all professionals in the company to review this architecture then give their questions (if any), then consider if this architecture is actually suitable for the current MPDC’s system. The data architecture should also ask the MPDC’s Technical team and business team that whether any hidden pain points are existed in system but not mentioned on the requirement document. Because those pain points if existed can change the whole new architecture.

Also, to save time, the CTO should consider that this architecture would be run on-premise or run on Cloud. If it runs on-premise, the CTO and the data architecture should discuss how the engine and hardware capacity should be set up, then to hire or to buy new one. If it runs on Cloud, the CTO and the data architecture should discuss what Cloud to use and the estimation cost of the whole system for a month or a year should be considered also.

# 9. References

**A. Apache Ecosystem:**

* + - 1. Apache Hadoop: <https://hadoop.apache.org/>
      2. Apache Nifi: <https://nifi.apache.org/>
      3. Apache Kafka: <https://kafka.apache.org/>
      4. Apache Spark (including Spark Streaming): <https://spark.apache.org/>
      5. Apache Hive: <https://hive.apache.org/>
      6. Apache Atlas: <https://atlas.apache.org/>
      7. Apache Cassandra: https://cassandra.apache.org/
      8. Apache Hadoop YARN: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
      9. Apache Druid: <https://druid.apache.org/>
      10. Apache Ranger: https://ranger.apache.org/
      11. Apache Knox: https://knox.apache.org/
      12. Apache Airflow: https://airflow.apache.org/

**B. BI Tools:**

1. Apache Superset: https://superset.apache.org/
2. Looker Studio: https://lookerstudio.google.com/

**C. Machine Learning frameworks/libraries**:

* + - 1. Tensorflow: <https://www.tensorflow.org/>
      2. Keras: https://keras.io/

1. PyTorch: https://pytorch.org/