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 |
| 03/12/2022 | Abhishek Kumar | 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**

***What?***

In the day and age when massive amounts of data are generated every single day, traditional systems, in most of the industries, are facing varied challenges. The absence of a well-established high-throughput architectural design for data systems can lead to major issues for an organization.

This design document addresses the current data challenges being faced by the *Medical Data Processing Company*, based out of San Francisco, which specializes in processing various types of Electronic Medical Records (EMR) and provides an enterprise data lake solution to their technical leadership.

***Why?***

*Medical Data Processing Company*, San Francisco, has experienced a surge in data and their existing infrastructure is not supporting this unprecedented rise in the volume and velocity of data. Their attempts at reconfiguring their hardware infrastructure have also reached its limitations. This document plan evaluates the company’s current system requirements and proposes an end-to-end enterprise data lake solution to solve the current and future challenges. I thank the Chief Technical Officer for reaching out.

The document contains the following:

* Requirements
* Data Lake Architecture Design Principles
* Assumptions
* Data Lake Architecture for Medical Lake Data Processing Company
* Design Considerations
* Conclusion
* References

***Target Audience***

This draft is addressed to the Chief Technical Officer, Chief Product Officer, Chief Data Scientist, Vice President of Data and Analytics, Principal Solutions Architect, Data Engineering Managers and other technical leadership of *Medical Data Processing Company*, San Francisco.

***In-scope elements***

* Remove data silos and create a one-stop solution that stores all the enterprise data centrally and enables teams within the organization to have seamless access to it.
* Scalable and sustainable data architecture that supports the increasing volume and velocity of data.
* Ability to create ad-hoc/custom reports, build dashboards on data visualization tools without moving the data from one system to another.

***Out-of-scope elements***

* The Internet of Things (IoT) introduces more ways to collect data, with real-time data coming from internet connected devices.
* Using social media analytics data to find most active customer cohort and provide them with promotions and rewards.
* Firewall and virtual network configuration of our scalable data lake architecture.

# **Requirements**

***Problem Statement***

*Medical Data Processing Company*, San Francisco, has witnessed rapid growth in data in the past years and is facing problems related to storage, scalability, data quality, schema flexibility and advanced analytics.

To address these issues, firstly, attempts were made to optimize their processes and logic that did not turn out to be a success. Secondly, the engineering team enhanced their hardware facilities and capacity but that did not solve the problem either. Today the data engineering team is dealing with server crashes, distributing data traffic to balance the load, non-responding systems due to failure, data duplication and unnecessary storage space and delays in restoring backups. All this has led to poor customer service as well as the formation of data silos throughout the company. At times when data analytics is crucial for businesses to optimize their performances and reduce costs *Medical Data Processing Company* is struggling with their analytics and reporting due to poor data management. Furthermore, they should be using their purging historical data to build Machine Learning models and near-real time dashboards but that is not happening as of now.

In conclusion, to solve all the existing data problems, the company needs a detailed layout of a Data Lake design and the justification behind this enterprise solution.

***Business Requirements***

The CTO plans to have a centralized data space that is scalable, reliable, efficient and fault tolerant. The data solution should lead to an automated data management system that improves their data ingestion processes, advances their execution times and overall enhances their customer experience. This is a perfect situation when an organization needs a data storage space for various end-users like business analysts, data scientists, product managers, executives, etc., to improve the business performance. A data lake can be leveraged to solve wide ranges of problems and provide solutions to the stakeholders.

***Summary of Current Technical Availability and Data Volume***

|  |  |
| --- | --- |
| **Current Technical Environment** | **Current Data Volume** |
| 1 Master SQL DB Server | Average zip files per day: 77,000 |
| 1 Stage SQL DB Server | Average data files per day: 15,000,000 |
| 3 Small Servers for Data Ingestion | 99% zip files ranges from 20 KB to 1.5 MB |
| Series of web and application servers | Data Volume Growth rate: 15-20% YoY |

# **Data Lake Architecture design principles**

In order to build a high-performance date lake that will serve the *Medical Data Processing Company’s* business requirements we need to consider some essential design principles prior to architecting our data lake.

***Design Principles***

* **Type of Storage**: Columnar-oriented formats allow data to be stored and retrieved in columns. This will help to read only the relevant medical data and result in huge performance benefits as well as high compression rates. Using columnar formats such as Apache Parquet and ORC helps us avoid scanning redundant data [1]. As the data volume of the *Medical Data Processing Company* is increasing by 15% year-on-year, supporting columnar formats will ensure that our queries run optimally.
* **Central Meta-Data Repository**: *Medical Data Processing Company* needs an architecture that has a centralized meta-data location which reduces their operational costs and other IT infrastructure resources. A central meta-data repository such as HBase consists of HMaster and HRegionServer. HMaster is responsible for managing all HRegionServers. It does not store any data itself, but only stores the mappings (metadata) of data to HRegionServer [2]. All nodes in the cluster are coordinated by Zookeeper and handle various issues that may be encountered during HBase operation.
* **Change Data Capture (CDC)**: Apache Hudi is an open-source data management framework used to simplify incremental data processing and data pipeline development. This framework more efficiently manages business requirements like data lifecycle and improves data quality. Apache Hudi allows us to manage data at the record-level to simplify Change Data Capture (CDC) and streaming data ingestion and helps to manage data privacy use cases requiring record level updates and deletes [3]. Apache Hudi can be operated on the Hadoop Distributed File System (HDFS) or cloud stores. Hudi’s two most widely used features are upserts and incremental pull, which give users the ability to absorb change data captures and apply them to the data lake at scale [4]. Hudi supports multiple query engine integrations such as Apache Hive, Apache Spark, and Apache Impala. Under the GDPR and CCPA laws *Medical Data Processing Company* needs to ensure how their customers data needs to be used. Hudi will aid in tracking as well as granular changes.

# **Assumptions**

Before we design a data lake architecture, it is important to discuss some assumptions.

* Our proposed data lake architecture embraces open-source tools in order to avoid proprietary solutions which can lead to vendor lock-in. As we are minimizing the use of cloud vendors, we presume that there will be no version mismatch and compatibility issues in our Hadoop stack.

*Potential Risk and Future Risk –Hadoop ecosystem tools will be relatively difficult to manage and will lack on-demand processing power. Auto-scaling clusters might be needed in future and not having pay as you go cloud vendor service the Medical Data Processing Company may end up paying more. We are not certain about the budget that the Medical Data Processing Company has allocated for this project*

* We assume that the Data Lake solution that we are providing to the *Medical Data Processing Company* does not lose its relevance long term. As a data architect, we will design a solution such that data and the metadata is easily searchable and discoverable across the organizations while keeping strong access control in place.

*Potential and Future Risk - As unorganized data without accurate metadata is ingested into the data lake, it can quickly flood the storage system with irrelevant data that clutters the system and makes retrieval complicated. If not managed properly, the data lake might end up becoming a data swamp.*

* With the medical data, there are state and federal rules that the company needs to abide by. As the *Medical Data Processing Company* is based out of California state law requires records to be accessible for up to 7 years.

*Potential Risk and Future Risk– Handling medical data is always challenging and the data governance team must keep with the changing rules and regulations. Ignoring such critical rules can cause the company risk of penalties.*

# **Data Lake Architecture for Medical Data Processing Company**

Diagram, engineering drawing

Description automatically generated

# **Design Considerations and Rationale**

## **Ingestion Layer**

***Data Ingestion Plan and Tools***

*Medical Data Processing Company* has data coming from the on-premises database servers, FTP servers and API extract agents.

Our plan is to use Apache Flume, Apache Kafka and Apache Sqoop to ingest FTP, real-time data from API and relational data respectively.   
Apache Flume is a data ingestion tool for transporting large amounts of sdata such as log files, events, FTP from various sources to a centralized data store.

We require a Linux virtual machine that should have the SSH service configured. Flume channels are the repositories where the flume events are staged on an agent.

In Flume channels, the sources add the events, and the sinks removes it.

Kafka is a high-throughput, distributed, fault-tolerant, and enterprise-ready event-streaming platform. Kafka’s main architectural components include Producers, Topics, Consumers, Clusters, Brokers, Partitions.

Finally, using a Sqoop job (Apache Sqoop) we will ingest on-premise relational data from the server to HDFS.

***Scalability***

Flume is an extensible, reliable, highly available, and horizontally scalable system. It is customizable for different types of sources and sinks. Apache Sqoop supports bulk import into HDFS. Sqoop parallelizes data transfer for optimal system utilization and fast performance. Kafka is highly scalable and in times of any node failure, Kafka allows for quick and automatic recovery. Kafka is excellent in data communication and integration as the volume of data increases.

***Other Tools and consideration***

Apache Airflow, Apache Nifi and AWS DataSync were other powerful and reliable data ingestion tools that were considered to move data from our data sources. Setting up Airflow architecture for production is not easy. A node cannot connect back to the cluster when disconnected accidentally unless admin manually copies the XML from the connected node. AWS DataSync, along with AWS CLI are cloud technologies for data ingestion but our proposed data lake architecture will embrace open-source tools as a business requirement.

## **Storage Layer**

***Storage Plan and Handling Growth of Data***

*Medical Data Processing Company* has an influx of structured data collected in SQL tables as well as a large amount of zip files (CSV, TXT, XML) and unstructured data arriving in all shapes and forms. Since this is clearly a problem of Big Data, we cannot collect all data in a single machine and run a query against it. The appropriate approach in this case should be to reduce I/O cost with minimum data movement as well as using multiple nodes running in parallel to operate on smaller datasets.

Considering a 20% YoY growth rate, we would use a distributed computing environment, Hadoop, to overcome the data volume and velocity challenges. Hadoop Distributed File System (HDFS) will enable fast data transfer among the nodes. We will install Hadoop on our engine for storage purposes.

***Backup and Metadata***

To safeguard our data in situations of crashes we will perform incremental backups. We could have selected third party storage vendors for this purpose, but we decided to use an Archive Cluster (HDFS Backup) – a backup dedicated HDFS cluster on a different physical location.

Allowing a central meta-data repository such as Apache HBase in our proposed data lake will aid customers to search and learn about the data sets in the lake. As mentioned earlier, HBase follows the master-slave architecture and consists of HMaster and HRegionServer. HBase will divide the logical table into multiple data blocks called HRegion and stores them in HRegionServer. HMaster is responsible for managing all HRegionServers. It does not store any data itself, but only stores the mappings (metadata) of data to HRegionServer [2]. All nodes in the cluster are synchronized by Zookeeper and handle various issues that may be encountered during HBase operation. This architecture will help the *Medical Data Processing Company* have a centralized meta-data location, reducing their operational costs and other IT infrastructure resources.

***Data Format and Data Security***

Columnar-oriented formats allow data to be stored and retrieved in columns. This will allow better compression rates as well as high performance at the time of reading the data. Apache Parquet and ORC are some popular columnar-oriented formats. With the increasing volume of data every year this will approach will help in avoiding scanning of redundant data.

All data stored in or accessible through HFDS is encrypted. Hadoop supports encryption at the disk, file system, database, and application levels.

In core Hadoop technology the HFDS has directories called encryption zones. When data is written to Hadoop it is automatically encrypted (with a user-selected algorithm) and assigned to an encryption zone [7]. Each file within the zone is encrypted with its own unique data encryption key (DEK). Clients decrypt data from HFDS uses an encrypted data encryption key (EDEK), then use the DEK to read and write data.

***Other Tools and consideration***

We could have used the components of AWS EMR Cluster which comes with Hadoop and HBase. In order to avoid vendor lock-in we ignored cloud solutions. Storage could have also been done on Amazon S3 buckets but we went with more open-source Hadoop ecosystem. A third option could have been Azure’s Data Lake Storage Gen1 which is a hyper-scale repository optimized for big data analytics workloads. It was not chosen for the similar reason.

## **Processing Layer**

***Tools, Processing Plan and enabling ad-hoc capabilities***

We choose Apache Hive and Spark for processing our data. Both are easy to setup and compatible with Hadoop stack. Hive is a declarative language like SQL, easy to write and can be queried by multiple users. This helps in solving ad-hoc querying needs for *Medical Data Processing Company.*

Apache Spark Streaming supports the processing of real-time data from various input sources and storing the processed data to various output sinks.  
PySpark is a Python API for Spark to support Python with Spark. The inbuilt API in Spark to operate data, makes it faster than pandas.

Apache Impala is another tool that we plan to leverage in order to process our data. Impala can be used to perform fast interactive analytics on unstructured data. For many types of queries, it's much faster than Hive and has a familiar SQL like language.

Apache Hudi simplifies Change Data Capture (CDC) and handles data privacy use cases requiring record level updates and deletes.

***Scalability***

Hive helps perform large-scale data analysis for businesses on HDFS, making it a horizontally scalable database. Apache Impala is capable of scaling to thousands of machines without sacrificing training stability or data efficiency. Spark is one of the most widely used engine for scalable computing as it is a bulk synchronous data parallel processing system.

***Drawbacks of our consideration, Other Tools***

We have not selected Apache Pig or Presto in our design consideration. Apache Pig is 36% faster than Apache Hive for join operations on datasets. Apache Pig is 46% faster than Apache Hive for arithmetic operations. Apache Pig is 10% faster than Apache Hive for filtering 10% of the data [9]. Apache Pig is not a good choice for pinpointing a single record in huge data sets as it is computationally expensive. Some of the customers need to pull individual records.

## **Serving Layer**

***What is Serving Layer?***

The serving layer is the last component of the data lake architecture and where the consumers of the data and other stakeholders get meaningful information from the once raw and crude data. This can be accomplished by various techniques that includes ad-hoc reporting, custom reports, data visualization, business intelligence, machine learning and deep learning models.

***Applications of Serving Layer***

With the help of Hive, we can process these data and serve our ad-hoc data requesters. Many complex requests and reports can be handled here.

The goal of any data driven business is to extract value from the data and help business to improve its day-to-day life cycle. One of the most popular ways to understand business is to convert these underlying data into visual forms by making graphs and charts to tell the story. Using Apache Impala, we make connection and setup the data source. Using an ODBC driver, we export our data extracts on Tableau and build stories. This is how we convert data into a proper story and extract meaningful value out of it.

PySpark is easy to use and is scalable too. It works on distributed systems. We can use Spark Machine Learning for various kind of data analysis. There are various techniques we can make use of with Machine Learning algorithms such as regression, classification, etc., with the PySpark MLlib. We can build, train and later deploy these models. With the rise in data, we can retrain our model as required.

# **Conclusion**

In conclusion, an enterprise data lake solution will help the *Medical Data Processing Company, San Francisco* to handle the rapid growth of data as well better analyze their data resulting in excellent customer service. This new vision will help them break data silos and make data available to the whole organization thereby saving time. It will allow them to scale horizontally and store data from diverse sources in its raw format. Also, it will help them solve complex queries and make machine learning and deep learning seamless.

# **8. References**

1. <https://www.udacity.com/course/data-architect-nanodegree--nd038>
2. <https://towardsdatascience.com/hbase-working-principle-a-part-of-hadoop-architecture-fbe0453a031b>
3. <https://aws.amazon.com/emr/features/hudi/>
4. <https://eng.uber.com/apache-hudi-graduation/>
5. <https://docs.aws.amazon.com/emr/latest/ReleaseGuide/UsingEMR_s3distcp.html>
6. <https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.6.3/bk_data-access/content/ch_hbase_bar.html>
7. <https://www.bmc.com/blogs/hadoop-security/#:~:text=Hadoop%20supports%20encryption%20at%20the,assigned%20to%20an%20encryption%20zone>.
8. <https://aws.amazon.com/blogs/machine-learning/build-amazon-sagemaker-notebooks-backed-by-spark-in-amazon-emr/>
9. <https://www.projectpro.io/article/difference-between-pig-and-hive-the-two-key-components-of-hadoop-ecosystem/79>

[10] Knowledge Hub, Udacity