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 |
| 06/06/2021 | Dotty Johnson | 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 |

# 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>

The Data lake Architecture design document is used to provide technical insights regarding the implementation of Data Lake Architecture solution.

The document contains in detailed information regarding the different components of the Data Lake, which tools are used to build those components and the reasons to choose those tools over other. The document is created for reference during the implementation of the data architecture as well as for future reference. The document is meant for the technical audience who needs information regarding the technical details and design ideas.

In scope elements:

* + Different layers and tools used
  + High Level Security strategies
  + Fault Tolerance strategies

Out of Scope elements:

* + Network Configuration for the Data Architecture.
  + Administration and Monitoring of Data Architecture
  + Tools used for security layer of data Architecture

# Requirements <approx. 1 page>

As the data volume continuous to grow due to company’s hyper growth, the current single node SQL server is not able to scale. The current solution has vertical scaling. The updating of

server hardware to maximum CPU, RAM and storage configurations is limited and cannot support huge data volume and requirement for increased processing speed. The system is not fault tolerant and doesn’t have a recovery plan. SQL server has a single point of failure and during a event of failure, restoring backup data takes hours resulting in poor customer services.

Due to limitation of capacity, for analytics, data is exported to separate servers on a nightly basis. This results in data duplication and wastage of storage spaces. There is also requirement to create ML models on historic real time data.

# Existing Technical Environment

* 1 Master SQL DB Server
* 1 Stage SQL DB Server
  + 64 core vCPU
  + 512 GB RAM
  + 12 TB disk space (70% full, ~8.4 TB)
  + 70+ ETL jobs running to manage over 100 tables
* 3 other smaller servers for Data Ingestion (FTP Server, data and API extract agents)
* Series of web and application servers (32 GB RAM Each, 16 core vCPU)

# 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

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

The requirements are obtained from the company profile problem statement document.

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

These are design principles used.

* Usage of horizontally scalable, distributed system to improve the performance by adding more instances/ nodes and to facilitate unlimited scaling to handle huge data volumes. Company is going through a hyper growth, resulting in rapid growth in data volume. Data Architecture solution should be able to handle these loads with failure.
* Strict security strategies as medical data hold sensitive personal data which few staff of the company can access to. Even though it is difficult to manage security in a horizontally scalable system, it cannot be compromised.
* Fault Tolerance and Lower Downtime: The system must be fault tolerant and in an event of failure, the system recover fast. This will ensure good customer service and well as continuous availability of data for the functioning of different department of the medical company.

# Assumptions <approx. ⅓ page>

These are the assumption used for designing HDFS.

* System will store data in large files.
  + Instead of data updates, data appends will be used.
  + Large sequential read
  + Small random reads

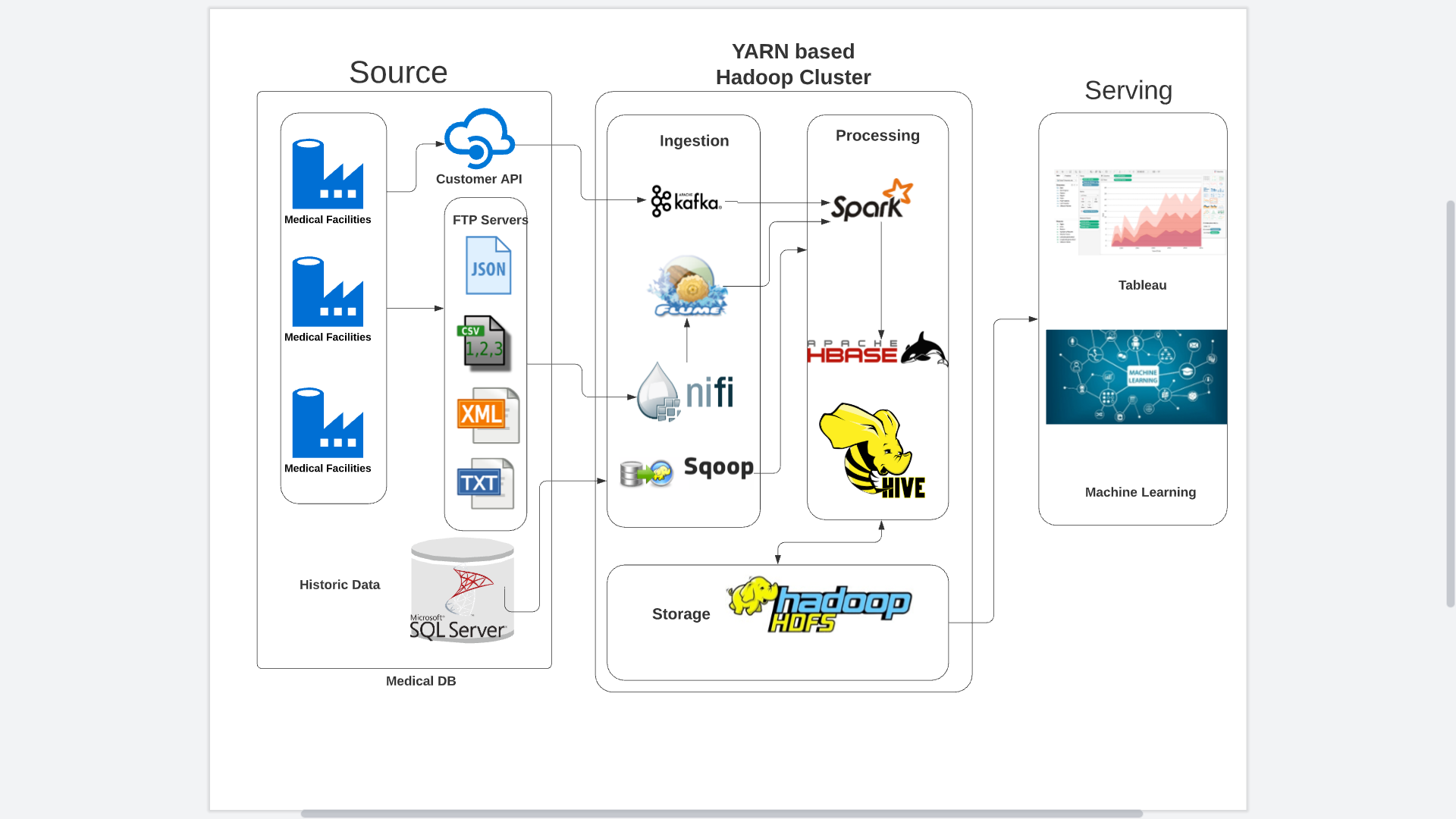
The type of data ingestion (data update or data append) is not mentioned in the problem statement. The HDFS is chosen based on the assumption it will data append instead of data update.

Another assumption is that large amount of data will be read sequentially,

Small amount of data will be read randomly.

If there are changes to the condition (e.g.: Data is changed for each record, then data append will not work. Data update will be required for data ingestion.), it will cause decrease in performance.

# Data Lake Architecture for Medical Data Processing Company



# Design Considerations and Rationale <at least 3 pages>

## Ingestion Layer

We have different data formats such as XML, TXT, CSV coming from FTP server. Company also uses a customer API to retrieve customer data. We also have a medical DB in MS SQL server for storing historic data. For ingesting all these different data types, we are using Apache Nifi, Apache Kafka, Apache Flume and Apache Scoop ingestion tools.

Apache Nifi will ingest the data from Customer API. NiFi processors are file-oriented and schema-less.

Apache Kafka will ingest the data from NiFi. Apache Kafka is open sourced and distributed system real time streaming data can be handled. It is horizontally scalable and is compatible with HDFS. Kafka provides **built-in scalability and fault tolerance** capabilities. If any of the servers fail in the cluster, the other servers will take over their work to ensure continuous operations without any data loss.

Apache Flume will be used to ingest data from FTP server. It is a distributed and reliable service for moving data of large quantity. Apache Flume is robust and Fault Tolerant. It also supports many failure and recovery methods. This ensures that moving data from FTP server to Hadoop cluster is more reliable and safer.

Apache Scoop will ingest data from MS SQL Server with historic data. Scoop is a good choice as it can transfer data between 2 databases.

Ingestion of data is important as it helps the data of different format/ sources enter the Hadoop Cluster where it could be processed and stored.

The ingestion layer will be horizontally scaled to ensure increased performance and infinite scaling.

AWS Kinesis is an option that I have considered. Kinesis is similar to Kafka in many ways.

The one disadvantage Kinesis has over Kafka is that it is a cloud service. This introduces a latency when communicating with an on-premise source compared to the Kafka on-premise implementation.

## Storage Layer

The data from the ingestion layer will be stororothY!12245

ed in HDFS. The data from Apache Flume and Apache Kafka will be processed using Apache Spark.

Processed data from Spark will be stored in HDFS. The file is broken into numerous chunks od data will be stored in the data nodes of the cluster. Data will be stored in columnar format for huge performance enhancement. It also offers higher compression rate that helps us to store vast amount of data.

The HDFS system is fault tolerant as it keeps multiple copies of same data. In case of a data node failure, another copy of same data in a different node will be used. If the master node fails, the stand by master node takes its place.

The meta data from HBase will be also stored in HDFS. The metadata contains information that enables CDC and UPSERT capabilities These are the information metadata will store.

* Index - Map rows to individual files
* Timeline metadata - tracks changes

The data should be secured using numerous methods

* + - 1. Restrictive access to the data. Certain data should be only available to certain staff of the medical facility.
      2. Column level masking. Sensitive data columns should be masked to avoid data leakage.
      3. Data Encryption at rest and transit.
      4. Checking what data is exposed via public endpoints and how through network level security.

The data will be stored be stored in Parquet format. Apache Parquet is a free and open-source format popular in processing Big Data workloads. Parquet uses snappy as the default data compression mechanism and provides column level compression to reduce storage space. Parquet allows queries to fetch/read specific column values that need not read the entire row data which significantly improves performance. Parquet provides better compression and encoding with improved read performance at the cost of slower writes.

Instead of Parquet, I also considered AVRO format. But since it has row oriented compression and Parquet’s compression is better that Avro’s compression, Parquet was chosen.

## Processing Layer

All the data from the ingestion layer (from Flume and Kafka) is pushed to Spark. Spark’s lightning-fast speed to process data through real-time and batch processing makes it an excellent choice. Spark can handle different types and structure of data. Spark Streaming enables data processing on both streaming and historic data. It can be perfectly clubbed with Apache Kafka. Spark SQL module is used for structured data processing. Spark uses deck scheduler along with query optimizer and a physical engine for both batch processing and real time data processing.

The Metadata catalog is stored in HBASE DB. HBase is a distributed scalable and reliable database

Hive is the data warehouse software used for ad-hoc querying. The megastore of Hive is used to store meta information of the tables ie. schemas and locations. Using CLI and Thrift Server, users can submit queries and monitor the results.

The processing layer will scale horizontally.

Apache Pig was initially considered instead of Apache Spark As Spark could handle both batch processing and real time streaming data and Pig doesn’t have that capability, Spark was chosen.

## Serving Layer

In the serving layer, we use the data for analytics solutions like Tableau and other data visualization tools or machine learning solutions

In the data visualization tools, data will be stored as structured data as data extracts that is queried from the processing layer of the data architecture. The data will be used to create power data visualization for data driven decision making.

In machine learning solutions, the data will be stored as data frames , arrays etc.

Using machine learning, we can build predictive models, anomaly detection models etc.

# Conclusion <approx 2-5 lines>

The Data Lake Architecture is a complete solution for managing increasing data volumes for the company. The data is ingested from multiple sources, processed, stored, and finally used to create data insights for the business.

# 9. References <If any>

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