Data Lake Architecture -

A Comprehensive Design Document

Medical Data Processing Company

By Richard A.

# Tracker

## Revision, Sign off Sheet and Key Contacts

## Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Change Reference |
| 06/04/2020 | FirstName LastName | 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 |

# 

# Summary

This document aims to cover the new Data Lake strategy for a proposed medical record processing company which we’ll call EMRP(Electronic Record Processing Company). Due to constraints with the current on-prem data architecture, EMRP has been dealing with many conflicts from database server crashes, to being offline for several hours for maintenance. This has become quite problematic for customers, stakeholders, employees and others. Also due to technological restraints and limited capacity of resources, data processing only occurs at night time. I am proposing EMRP move from a data warehousing architecture to a data lake architecture.

What is a Data Lake?

A data lake is a centralized data repository system that can store a wide array of data files and formats. They can store everything from structured, semi-structured, unstructured and even raw data files them selves without worrying about setting schemas. You can also build a datawarehouse within a data lake system should you need to. They are also flexible enough to support many different kinds of frameworks including but not limited to processing, ingestion, orchestration, analytics and more! It is the most cost-effective, robust, scalable solution to meet our problem.

Why a Data Lake?

A data lake provides a way to scale or simplify our data system architecture in a much more convenient manner. The current datawarehouse, single-node architecture isn’t enough to grow with scale despite all the optimizations made allowing it to be as performant as possible. Even with these optimizations the data warehouse, single node system was crashing and had several hours of downtime. These are risks the company cannot afford to make as it will drive customers away to competitors.

In-Scope:

Three in scope elements are the data ingestion framework, data storage architecture and processing/analysis. For data ingestion we need the most robust tools for seamless batching and streaming and for this we are proposing to use Apache Nifi for batching and Apache Kafka for streaming. We also established that the data architecture needs to be scalable, and cost-effective. Using Apache HDFS will help aid in storing all the data, including raw data as well. For our metadata needs we’ll use Apache Cassandra for supporting the metadata storage. Lastly we’ll use Apache Spark for our big data cleaning and processing needs before serving the data to our end users on the tableau dashboards and other tools to be queried by our end users.

Out of Scope:

One thing to consider is if we need to pull data from legacy systems as this wasn’t discussed in the requirement. If so we’d also need to build a customized pipeline to pull in that data. Dealing with legacy systems can be a super complex ordeal. Another element to consider is application development, although we proposed to use Tableau for dashboarding purposes there maybe other tools to consider connecting to that was part of the old data warehouse system. Lastly we’d need to consider training the end-users on how to use the new data lake system as this wasn’t discussed in the requirements either.

Requirements:

# Current Data Volume:

* Data coming from over 8000 facilities.
* 99% zip file size ranges from 20kb to 1.5mb.
* Edge cases: some large zip files are as large as 40mb.
* Each zip file when unzipped will provide either CSV, TSV, XML records.
* Incase of XML zip files, each zip file can contain anywhere from 20-30 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: 3,500.
* Average data files per hour: 700,000.
* Data Volume Growth Rate: 15-20% YoY

# Business Requirements:

After meeting with the stakeholders, a list of requirements were compiled to go over the necessary hardware needed to scale to accommodate for the massive growth in data. The following requirements are:

* Improve uptime of the overall system through fault tolerance and reliability.
* Reduce latency for SQL queries and reporting.
* Architecture should be able to scale with the volume and veracity of data.
* Improves business agility and speed of innovation through automation and ability to experiment new frameworks.
* Use open source tools and avoid proprietary solutions. This can mitigate 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 data source.
* Centrally store enterprise data and enable easy access.

# Technical Requirements:

* Ability to process incoming files on the fly instead of night time batch loads today.
* Separate the metadata, data and compute/processing layers.
* Ability to keep unlimited historical data.
* Ability to scale up processing speed with increased data volume.
* System should sustain a 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 the 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 like PowerBI, or Tableau.
  + Generate daily, weekly, nightly reports using scripts or SQL.
* Ad Hoc-analytics, interactive querying capability using SQL.

# Data Lake Architecture Design Principles:

* Scalability:
  + Ensure new data lake architecture can scale to support rapid data growth.
    - This is to ensure that the new data lake system doesn’t run into the same compromise on performance to eliminate downtime and allow for processing throughout the day instead of just the night time.
  + Optimizing budget efficiency by leveraging open source tools where applicable.
    - By using open source tools, we can allocate paid resources on other things.
  + Use distributed storage and processing systems such as AWS S3 for storage and Apache Spark/Pyspark for parallel processing.
* Reliability & Fault Tolerance:
  + The system needs to be operational despite potential future failures.
  + Adding data replication, backup strategies, and using frameworks such as Apache Nifi & Kafka mitigates any fault tolerant issues that may arise.
  + This ensures the reliability of critical and real-time medical data at all times while efficiently processing new incoming data and minimizes any potential hardware/software failures.
* Data Security & Compliance:
  + System must protect sensitive medical data and comply with HIPAA regulations.
  + Use encryption and conduct scheduled audit tools like AWS Key Management and IAM roles.
  + This allows for protecting patient privacy and anonymity while also avoiding potential legal issues.
* Cost Efficiency:
  + System should provide value while minimizing costs.
  + Can do this by using tiered storage solutions, and serverless architectures where applicable.
  + This helps ensure sustainability in the system allowing for additional future innovations.
* Performance:
  + Must provide timely access to data in order to support real-time analytics and reporting.
  + Process data in-memory using tools like Apache Spark/Pyspark and optimize data storage formats by parquet, ORC etc…
  + High performance enables rapid retrieval and processing of data. This is a must for real-time insights and decision making in important medical situations.
* Flexibility and Extensibility:
  + Allowing the system to adapt to new data sources, and evolving requirements.
  + Design modular components, using APIs and microservices architecture.
  + This allows the system to grow and evolve when needed especially when incorporating additional use cases.

# Assumptions:

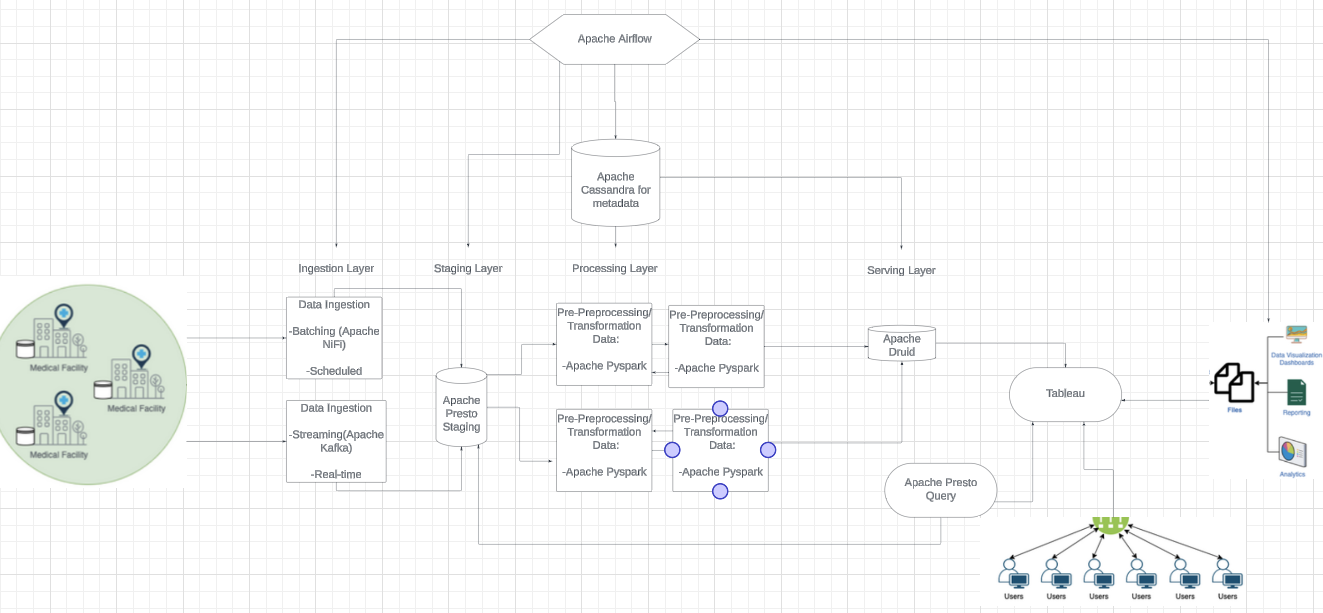
* What is the current and projected data volume for the 8000 medical facilities?
* Are there any semi-structured, or unstructured data that needs to be considered to support or improve in dashboard building, reporting, and analytics?
* Are there any legacy systems that need to ingest data from?
* As data volume and veracity fluctuates, choosing the appropriate storage solution are processing frameworks is crucial.
* Ex. Apache HDFS, Apache Kafka/Nifi and Apache Pyspark.
* Are there different user groups that I need to set up for account and access purposes?
  + What are the specific needs for these user groups?
* Integrating existing EMR systems and other infrastructure might be more complex than initially thought. Test and use standardized APIs for smoother integration.

# Potential Current and Future Risks:

* Data quality might be an issue if the proper data governance methods aren’t put in place.
* Storing sensitive medical data in the data lake system can be risky as it poses a threat for any potential data breaches in the future if not secured properly.
* Integration complexity, as this wasn’t discussed in the initial business requirements, it would be another thing to consider when integrating older systems with the new proposed data lake system.
* Some future risks to consider:
* Would be the rapid evolution of technology as tech stacks are always changing which may require future adoptions of new tech.
* Cost management is another thing to worry about, as unmonitored billing can cause escalated costs over time and going over budget.

# 

# Data Lake Architecture for Medical Data Processing Company:



# Design Considerations and Rationale <at least 3 pages>

1. Ingestion Layer:
   1. How do you plan to ingest different types of data?
      1. The system can use a combination of batch processing and stream processing, API integration, and file transfer methods.
      2. This helps ensure that multiple data sources are covered, including structured, semi-structured(from APIs), and unstructured data from files.
      3. Use Apache NiFi.
      4. Apache NiFi allows you to pull in data from databases, apis, and files.
2. How to scale ingestion layer:
   1. Adding more instances or nodes of apache nifi, allows for ingestion to scale horizontally. As data volume grows, the system can handle increased loads without any hindrances on performance.
3. Other Tools?
   1. Apache Cassandra: To allow for log data ingestion to save in a metadata database.
   2. Apache Airflow: For task orchestration and automating workflows.
4. Storage Layer:
   1. How to store vast amounts of data?
      1. Primary Storage: Apache HDFS to hold all raw data files post ingestion. It’s scalable, durable, and cost-effective.
      2. Apache Cassandra for metadata.
5. Processing Layer:
   1. ETL Tools: Apache Spark for data processing and transformation.
6. Serving Layer:
   1. Querying: Apache Druid as an acting data warehouse for structured data.
   2. Apache HDFS for SQL queries on raw data files stored in Presto.
   3. Data Visualization and Dashboarding: Tableau

# 

# Conclusion

This new data lake architecture solution should enable Company EMRP to scale their business properly devoid of bottlenecks, system crashes and critical downtimes. Incorporating a balance of open source technologies and paid technologies, allows EMRP to optimize their budget using the most modernized tools on the market.