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 |
| 08/05/2024 | Phuong Cao | 0.1 | Initial draft |

## Reviewers / Approval

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

## Key Contacts

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Role | Team | | email |
|  | Data Architect | | Medical Data Processing |  |

Table of Contents

1. Purpose
   1. Summary
   2. Target Audience
   3. In-scope
   4. Out of-scope
2. Requirements
   1. Requirements of Data Lake solution
   2. Existing Technical Environment
   3. Current Data Volume
   4. Business Requirements
   5. Technical Requirements
   6. Where do you find these requirements? Have you seen them somewhere before?
3. Data Lake Architecture design principle
4. Assumptions
5. Data Lake Architecture for Medical Data Processing Company
6. Design Considerations and Rationale
7. Ingestion layer
8. Storage layer
9. Processing layer
10. Serving layer
11. Conclusion
12. References

# Purpose

1.1 Summary

This material is the design document that provides the Data-lake Architecture Design solution proposal for the “Medical Data Processing” company’s current problems. The Company has experienced hyper growth over the past 3 years and current technology stack cannot catch up with the volume of data increasing and spike issue that lead to a single point of failure on server that hosting critical customer data. The CTO desired the data lake solution will address the company’s current and future challenges.

1.2 Target Audience

“Medical Data Processing” company :

- C-level

- Data Engineers and Database Administrators

- Data and Business Analysts

- Data Scientist

1.3 In-Scope Items

- Requirements, Assumptions, Design of the data architecture

1.4 Out-of-Scope Items

- System Implementation, Data Governance and Stewardship, Machine Learning

# Requirements

* 1. Requirements of Data Lake Solution

The Data Lake Architecture should provides:

- The High availability, reliability and resiliency system.

- The system easily in scale that meet the current growth in data volume and processing performance.

- Removing the data silos within the company and provide the single source of truth.

* 1. 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)

* 1. 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

* 1. Business Requirements

- Improve uptime of overall system

- Reduce latency of SQL queries and reports.

- System could be reliable and fault tolerance.

- Architect should scale as data volume and velocity increase.

- 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 be lead to vendor lock-in.

- Meta-data driven design - a set of common scripts should be use 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.

* 1. Technical Requirements

- Ability o process incoming files on the fly ( instead of nightly batch loads today).

- Separate the meta-data, data and compute/processing layers.

- Ability to keep unlimited historical data.

- Ability to scale up processing speed with increase in data volume.

- System should be 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 the same data-set, 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.

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

# Data Lake Architecture design principles

Data lakes are the cornerstones of modern big data architecture, the four essential design principles that help us for building the right and effectively data lake architecture.

* **“Event sourcing”** to ensure data traceability and consistency. The approach is “store now, analyze later”

1. The first step is to maintain  an immutable log of all incoming events on object storage, like Amazon S3, Azure blob storage ..
2. Next, implement ETL processes to extract, transform and load them consumption platform.

The event sourcing enables the ETL processes will be logging to the immutable log. If there was an issue in your ETL code, you can easily fix it and run the new code on the immutable original data.

* **“Layer data lake”** when we [organize our data lake](https://www.upsolver.com/blog/best-practices-to-organize-your-data-lake-and-drain-the-data-swamp), we have the possibility to store multiple copies of the data for different use cases and consumers by automating the ETL pipelines that ingest the raw data and perform the relevant transformations per use case.
* **“Keep the architecture open”** that mean avoiding vendor lock-in or proprietary on specific tools or solution. The best design for the data lake that central storage organizational data in one place under open of data type such as: ORC, Apache Parquet .. while also enabling the access to the data with a wide variety of tools, and by a broad range of services.
* **“Plan for performance”**

# 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

# 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?>

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

<What tools would be used? Why? >

<How would the ingestion layer design scale?>

<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?>

## Storage Layer

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

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

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

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

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

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

<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?>

## Processing Layer

<How do you plan to process the data?>

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

<How do you enable ad-hoc querying capabilities?>  
<What different tools are involved for processing?>

<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?>

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

## Serving Layer

<What do you mean by serving layer?>

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

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

# 8. Conclusion <approx 2-5 lines>

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

# 9. References <If any>

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