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 |
| 13/05/2023 | Kien Dang | 0.1 | Initial draft |

## Reviewers / Approval

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Version Approved | Position | Date |
|  |  |  |  |

## Key Contacts

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Role | Team | email |
| Kien Dang | Data Architect | Medical Data Processing | kiendt20@fpt.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

The purpose of the document to provide detail technical design proposal for an enterprise data lake system.

The document contains a detailed technical design proposal for an enterprise data lake system. It’s includes descriptions of the architecture, explanations of how the proposed design can solve the company’s challenges. And document should clearly state any assumptions or potential risks to the design.

The document is intended to showcase data architecture ability to design a technical solution for an enterprise data lake system and to demonstrate understanding of the business problem and recommendation for a solution.

This document is aimed at technical audience who is interested in your design ideas and decisions at a deep level.

In scope: a design of data lake system architecture that can handle large volumes of data, integrating various data sources, providing a flexible and scalable solution that can meet computer’s needs, a detailed explanation of the proposed solution, including the technologies and tools used.

Out of scope: implementation of the data architecture, data governance, machine learning.

# Requirements

Summary:

* Design a system with high availability, reliability, resiliency.
* Easily Scalable with increasing in data volume and velocity.
* Ad-hoc data analytics, interactive querying capability using SQL.
* Integrate flexibly with report, dashboards, ML frameworks.

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 & 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: 3,500**
* **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
* 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

Where do you find these requirements: From company profile problem statement file.

# Data Lake Architecture design principles

**Scalability & Flexibility**: Medical Data System will be dealing with large amounts of data, and as the amount of data grows, the system needs to be able to scale up or down as needed. And with dealing with different types of data, including (CSV, TXT, XML . . .). By designing a flexible architecture, Medical Data Systems can ensure that their data lake can handle any type of data that they may encounter.  
But when change format file from CSV, XML, TXT to Avro and Parquet which are standard, well-known and can accessible by different tools may good for performance.

**Resilience**: Use a central meta-data repository such as AWS Glue. This will allow data architecture to centralize and manage all meta-data in a single location, reducing operational costs in infrastructure, IT resources and engineering hours. Event sourcing should be used and an immutable log of all incoming events and maintained on object storage. Event sourcing enables data engineer to retrace the steps to learn about the extract transformation applied on the raw data.

**Easy to use**: By automating the ETL pipelines that ingest the raw data and perform the relevant transformations per use case we can prevent the data engineering bottleneck that might form if relying on coding-based ETL frameworks such as Apache Spark.

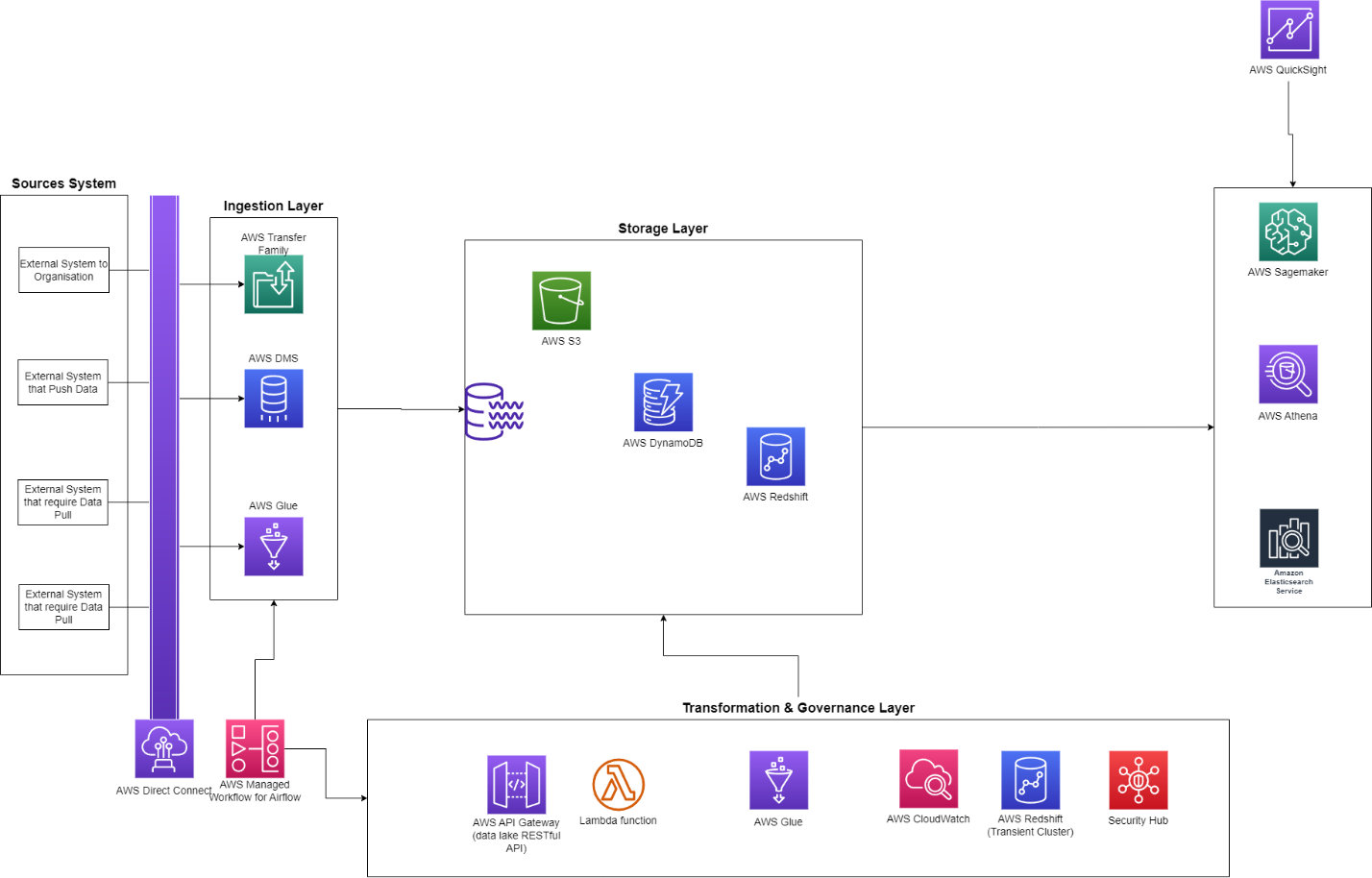
**Performance:**

* Every file stored need contain meta-data to understand data structure.
* Use columnar file formats: Apache Parquet and ORC
* Keep data in optional file sizes, A hot/cold architecture is recommended like hot for frequency files, cold for un-frequency files
* Use efficient partitioning strategy.

# Assumptions

* Time to switch to data lake as soon as possible
* 100% of data will be moved to the cloud
* Here it is possible to build a data lake on premise but I want to build architecture in the cloud. The goal of building a data lake system architecture in the cloud can help scale quickly, not take much time to maintain, many services will be automated and no warranty fees.
* Cloud is preferred over on-premise infrastructure. An on-premise data lake imposes challenges. Companies must build their own data pipelines, pay the ongoing management and operational costs in addition to the initial investment on servers and storage equipment, and manually add and configure their servers to scale a data lake to cater to move users or increasing data volume.

# Data Lake Architecture for Medical Data Processing Company



# Design Considerations and Rationale

## Ingestion Layer

Amazon has several tools that can ingest data to S3, Redshift, DynamoDB.

* AWS Direct Connect is a network service that provides an easy way to move data files from the applications to S3. This reduces the chance of hitting bottlenecks or unexpected increases in latency. AWS Direct Connect makes it easy to scale connection to meet your needs
* AWS Transfer Family imports hundreds of terabytes of data quickly into AWS using Amazon-provided secure appliances for secure transport. Multiple devices can be used in parallel or clustered together to transfer petabytes of data into or out of AWS.
* AWS DMS is cloud service that makes it possible to migrate relational databases, data warehouses, NoSQL databases and other types of data stores. You can use AWS DMS to migrate data into AWS Cloud or between combinations of cloud and on-premises setup and AWS DMS support CDC tasks.

Some open sources tools I considered that: Apache Sqoop, Flume, Kafka, Debezium, Nifi but I decided use Amazon services:

* Sqoop is a tool designed to transfer data between Hadoop and relational database servers. It is used to import data from relational databases such as MySQL, Oracle to Hadoop HDFS, and export from Hadoop file system to relational databases. This is a brief tutorial that explains how to make use of Sqoop in Hadoop ecosystem.
* Flume is a tool/service/data ingestion mechanism for collecting aggregating and transporting large amounts of streaming data such as log files, events (etc. . .) from various sources to a centralized data store. Flume is highly reliable, distributed, configurable tool. It is principally designed to copy streaming data (log data) from various web servers to HDFS.
* Kafka is designed for distributed high throughput systems, can handle a high volume of data and enables you to pass messages from one end-point to another. Kafka is suitable for both offline and online message consumption. Kafka messages are persisted on the disk and replicated within the cluster to prevent data loss.
* Debezium is an open source distributed platform for change data capture. Debezium lets your apps react every time data changes, and you don’t have to change your app that modify the data. Debezium continuously monitors your databases and lets any of your applications stream every row-level change in the same order they were committed to the database.
* Nifi is a realtime data ingestion platform, which can transfer and manage data transfer between sources and destination systems. It supports a wide variety of data formats like logs, geo location data, social feeds, etc. It also supports many protocols like SFTP, HDFS, Kafka . . . This support to wide variety of data sources and protocols making this platform popular in many IT organizations.

## Storage Layer

For data storage, the data lake architecture is based on Amazon S3 as primary data store. Amazon S3 offers a range of storage classes designed use cases. For example, you can store mission-critical production in S3 Standard for frequent access, save costs by storing infrequently accessed data in S3 Standard-IA or S3 One Zone-IA, and archive data at the lowest costs in S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive. Amazon S3 doesn’t have any limits for the number of connections made to the bucket. I recommend store data in open formats such as Apache Parquet, ORC that are standard, well-known and accessible by different tools. The metadata can be populated by AWS Glue, the web interface or via API, and store in DynamoDB. Data can be shared with multiple data lakes through the S3 buckets. Data can be replicated in different regions for back-up recovery. Redshift is very good choice for SQL-based access. Redshift is a fast and fully managed petabyte-scale data warehouse that costs less than $1000 per terabytes per year. The Redshift cluster can be resized to change the node type, number of nodes or both.

With open sources tool, I suggest Apache Hadoop HDFS (Hadoop Distributed File System) provides a distributed file system that is designed to run on commodity hardware. It has many similarities with existing distributed file systems. It is highly fault-tolerant and is designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications have large datasets.

With 20% YoY Data Growth rate, S3 can handle to scale this. Spreading requests across many connections is a common design pattern to horizontally scale performance. Data will be stored I the native formats upon ingestion. After ETL, data can be stored in open formats such as Apache Parquet and ORC as standard, well-known and accessible. by different tools.

I considered the cloud provider from Microsoft and Google. But Amazon S3

## Processing Layer

For processing and analysis, the services from AWS will be used, they have built-in scalability.

* AWS Lambda is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers. Use cases of Lambda are: process data at scale, run interactive web and mobile backends, enable powerful ML insights, create event-driven applications.
* AWS Glue is a serverless tool developed for the purpose of extracting, transforming, loading data. This process is referred to as ETL. AWS Glue enables businesses to extract data from one source, transform the data, load it into a data warehouse, all from the cloud. AWS Glue is also a fully managed service, which means we as users don’t have to manage any cloud infrastructure, it’s all taken care of by Amazon. It’s integrated with S3, Redshift, other JDBC compliant data sources and auto suggest schemas and transformations, which improve developer productivity. You can also view and edit the code of ETL, can schedules the ETL jobs and auto-provisions and scales the infrastructure based on the job requirements.
* EMR (Elastic MapReduce) is a fully manged service that make it easy to process large amounts of data using open sources tool such as Apache Spark, Apache Hadoop and Hive. It provides a managed Hadoop framework that allows you to easily process and analyze large amounts of data using a distributed computing model. EMR allows you to easily provision, configure, and managed a Hadoop cluster in the cloud.

Some open source a suggest Apache MapReduce, Pig, Hive, Spark, as the processing tool.

* MapReduce is a programming model and an associated implementation for processing and generating large data sets. It is used to process large amounts of data in parallel across a large number of nodes in a cluster. The MapReduce programming model consists of two main functions: Map and Reduce.

The Map function takes a set of data and converts it into another set of data, where individual elements are broken down into key-value pairs. The Reduce function then takes the output of the Map function and combines the data with the same key. The output of the Reduce function is a set of key-value pairs where the values are aggregated based on the key.

MapReduce is widely used for processing large amounts of data in distributed systems. It is used in many big data processing frameworks such as Hadoop, Apache Spark, and Amazon EMR.

* Pig is a high-level platform for creating MapReduce programs used with Hadoop. It provides a simple language called Pig Latin, which is used to write scripts for data analysis. Pig Latin is a procedural language that allows you to express data transformations using a set of operations such as filtering, grouping, and sorting.

Pig Latin scripts are compiled into MapReduce jobs, which are then executed on a Hadoop cluster. Pig provides a number of built-in functions and operators that make it easy to perform common data transformations.

Pig is widely used in data processing pipelines for ETL (Extract, Transform, Load) operations, data cleaning, and data analysis. It is often used in conjunction with other big data processing frameworks such as Apache Spark and Apache Hive.

* Hive is a data warehousing framework built on top of Hadoop. It provides a SQL-like language called HiveQL, which allows you to write queries that are translated into MapReduce jobs. HiveQL is similar to SQL, but it is optimized for querying large datasets stored in Hadoop.

Hive provides a number of built-in functions and operators that make it easy to perform common data transformations. It also supports user-defined functions (UDFs), which allow you to write custom functions in Java or another programming language.

Hive is widely used for data warehousing and data analysis. It is often used in conjunction with other big data processing frameworks such as Apache Spark and Apache Pig.

## Serving Layer

The serving layer is a layer in a big data architecture that is responsible for serving data to end users or applications. It is the layer that provides low latency access to data for real time applications. The serving layer typically consists of a distributed database or key-value store that can handle high volumes of read and write requests.

Some key services include:

* Amazon S3: Amazon S3 is a highly scalable object storage service that can be used to store and serve large amounts of data. It is often used as a data lake for storing raw data that can be processed by other services in the big data architecture.
* Amazon DynamoDB: Amazon DynamoDB is a fast and flexible NoSQL database service that can be used to store and serve structured data. It is often used for real-time applications that require low-latency access to data.
* Amazon Elasticsearch Service: Amazon Elasticsearch Service is a managed search and analytics service that can be used to search, analyze, and visualize large amounts of data. It is often used for log analysis, security analytics, and business intelligence.
* Amazon Kinesis: Amazon Kinesis is a real-time data streaming service that can be used to ingest and process large amounts of data in real-time. It is often used for real-time analytics, machine learning, and IoT applications.

# 8. Conclusion

To summarize, below is the architecture of the proposed data lake:

* For storage, using Redshift, S3, DynamoDB for secure, cost-effective, durable, and scalable storage.
* Data can be quickly and easily ingested into S3 from SQL Server, FTP Server, APIs by Direct Connect, AWS Transfer Family, Glue, DMS.
* For processing and analyzing data, EMR, Redshift, Athena, and other AI services, so any analytical solution can rapidly scaled to power any big data applications.

# 9. References