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
2022/06/05	Juan Barbosa	0.1	Inital draft

Reviewers / Approval

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

Key Contacts

Name	Role	Team	Email
Juan Barbosa	Data Architect	Data	juan.barbosa@zinobe.com

Contents

1 Purpose

The rapidly growing volume of data the company has collected over the past 3 years has put pressure on the current infrastructure. Optimizations have been made as well as configuration changes, and hardware updates, but performance and scale are still not within expectations.

This is the comprehensive design document with a proposed data lake architecture, describing an architecture that addresses these problems.

It includes the designed end to end data architecture, technical requirements, assumptions and risks we might encounter when implementing the data lake solution. The proposed architecture is expected to be complemented with data governance guide lines, a budget estimation, and examples of the ingested, processed, stored and consumed data, although they are not withing the scope of this document.

This document is intended for enterprise architects, software engineers, and technical directors.

Context

Medical Data Processing Company, specializes in processing various types of Electronic Medical Records. About 8000 medical care facilities use the product to:

- stay compliant with laws
- track patient health metrics
- admit discharge records
- bed availability

Such facilities that use these real time insights are:

- Urgent Care (UC)
- hospitals
- nursing homes
- emergency rooms
- critical care units

2 Requirements

2.1 Existing Technical Environment

- 1 Master SQL DB Server
- 1 Stage SQL DB Server
  - 64 core vCPU
  - 612 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)

2.2 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
- Data Volume Growth rate: 15-20 % YoY
- Average:
  - Zip files:
    - per day: 77,000
    - per hour: 3,500
  - Data files:

    - per day: 15,000,000
    - per hour: 700,000

2.3 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

2.4 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

3 Data Lake Architecture design principles

The architecture design principles are set according to *4 Guiding Principles for Modern Data Lake Architecture*[1]. They are the following:

3.1 Event sourcing

First step is to store data as is, by doing this there is full traceability inside the company infrastructure. Furthermore, if there are any changes in how the data is processed there is always the possibility to reprocess historic data.

After RAW data is stored, ETL jobs analyze data and load the resulting data to the storage layer to be used by consumer layers.

3.2 Data lake according to your users skills

Having multiple users means multiple interests in how data is used. ETL can ingest the raw data and transform it based on the different use cases of each customer. Even if the source data is the same, the Data Lake storage layer can store multiple transformed copies, reducing the computational impact of calculating each case when data is queried.

3.3 Open architecture

Using a data lake to store historical data is often cheaper than data stored in databases. Nevertheless, use open formats to store data, not proprietary or specific formats.

3.4 Plan for performance

- Metadata is needed to understand the data structure. Without the under laying data structure, query engines like Amazon Athena will not work
- By using columnar file formats, speed increases as there we avoid scanning redundant data from unused columns
- Files should have an optimal size
- An efficient partitioning strategy also helps avoiding unnecessary data to answer an specific analytical question

4 Assumptions

There are some assumptions that were made for this proposal:

- **Technical knowledge:** the proposed solution relies on AWS infrastructure, which requires some knowledge to use
- **On cloud:** the cost of an on premise infrastructure is much higher to start with. Furthermore, the current SQL DB servers can not be used as on-premise until data migration and the new architecture is implemented, meaning that new technical environment must be acquired.
- **Single cloud:** the whole architecture is based on AWS, since is cheaper to keep everything inside the same cloud. Nevertheless, there are other cloud providers that could host the same architecture (Google Cloud [2], Azure [3]).
- **Bulk load:** a bulk load of data is required in order to migrate data from the SQL servers into the AWS S3 buckets.
- **Data Governance:** data governance will be addressed as part of the implementation of the Data Lake.
- **Apache Spark Knowledge:** ETL processes are planned to be run using Apache Spark inside an AWS EMR environment, such processes need to be implemented by a team of data engineers.

There are at three concerns of the current proposal:

- **Migration:** data from the current SQL infrastructure must be migrated into the processed entries of the data lake, but moving 8.4 TB of data is no small task. Furthermore, if the service is running, there must be a bulk load, followed by incremental changes from the start of the procedure. If this process is not sync correctly, there is the potential for data loses.
- **Security:** high security of the S3 buckets must be enforced, if security profiles are not correctly configured there could be unwanted access to the lake.
- **Data Quality:** Since sources are not monitored, and are to be automatically uploaded to the data lake, there might be issues of data quality if the structure of the incoming data changes.

5 Data Lake Architecture for Medical Data Processing Company

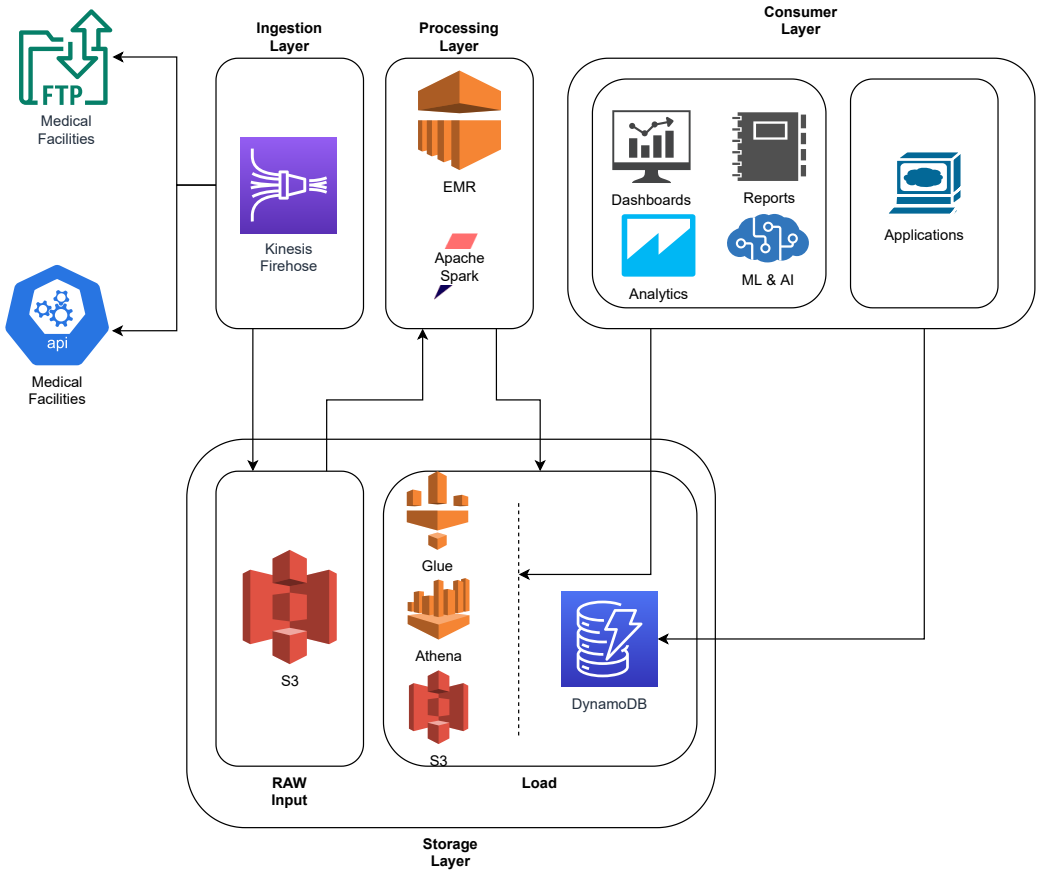

Figure 1: Proposed architecture diagram

6 Design Considerations and Rationale

6.1 Ingestion layer

There are three types of files to be ingested: `XML`, `TXT`, `CSV`, all of which are compressed using `gzip` format and password protected. Two major sources of information are mapped: FTP servers and uploads through API from the

Medical facilities. The ingestion layer is made with *Amazon Kinesis Firehose* [4] service, having the following benefits:

- near real time
- fully managed
- can handle any amount of streaming data

The first and last are technical requirements. The middle one is a facility, keeping in mind that the current architecture is already having trouble with the handling of the data, the implementation of the proposed architecture should be as soon as possible. A tool like Amazon Kinesis Firehose is perfect for the task, as there is no need to manage intermediate services and infrastructure.

There were some open source tools considered:

- **Google PubSub:** used for asynchronous messaging between applications, thus, not suitable for the task [5].
- **Apache Nifi:** used between software systems, thus, not suitable for the task [6].
- **Apache Kafka:** distributed event streaming platform, suitable for the task, but not managed [7].
- **Apache Sqoop:** used to copy from relational data sources into Hadoop. Eventhough a relational database might be a source of data from the medical facilities, it would not handle API, nor FTP. Furthermore, as of today, is a retired project [8].

6.2 Storage layer

The storage layer is powered by Amazon S3 service. S3 can grow indefinitely, thus, it can handle any amount of data, with any growth rate [9]. In order to fulfill with the event sourcing principle of the data lake architecture design, raw data is stored in one bucket in S3. Processed data is also to be stored in S3, using a Parquet file format, with the primary purpose of serving machine learning and analytic teams. In case of modification, or recovery of the processed data, the raw data S3 bucket serves as back-up. Furthermore, if

needed S3 buckets can be replicated along other AWS regions for further back-up.

Parquet file format is used because is column oriented, and it is almost and industry standard that allows for schema evolution, compression, splitability, and finally it is optimized for read operations. Since this are the processed entries, read is preferred over write.

In order to support for the Ad-hoc SQL queries, Amazon Athena is used [10]. Amazon Glue Data Catalog is used to store the data lakes meta data, allowing for classification, lineage, and discovery [11, 12].

S3 security will be based on:

- not publicly accessible
- Virtual Private Clouds are required

Eventhough there is a great open source tool for the job: the Hadoop Distributed File System (HDFS)[13], as explained in the ingestion layer this is a not managed solutions, thus, more complex to implement and maintain.

Finally, a DynamoDB [14] is also implemented to have the least latency in the data required by the applications. On this database only the most frequently queried data is available for the application to retrieve.

6.3 Processing layer

The processing layer is made with an Amazon Elastic Compute Cloud (EMR, [15]) with Apache Spark implemented. Apache Spark is the fasted option for MapReduce tasks, even compared to other Apache solutions (Apache Hadoop and Apache Pig, [16, 17]).

Since Apache Spark is a distributed computation system, it can be scaled horizontally to keep with the increase in data size [18]. Furthermore, Spark allows the processing of data in batch and real time, and by reading entries from the loaded S3 parquets, CDC.

Any automatic transformations of data are programmed using any of the supported programming languages of Spark, and the output is loaded back into S3 as parquets files, and most frequently used data by the applications are loaded into DynamoDB.

Jobs can be scheduled using `cron` inside EMR. An instance of Apache Airflow could be used to orchestrate the tasks if the number of jobs increases in volume an complexity [19].

6.4 Serving layer

The service layer is composed by two major areas:

- **Applications:** Represents the flow of database requests from the application layer in the 3-tier application architecture. These requests are directed to the DynamoDB in the storage layer, giving low-latency results and auto scaling capabilities depending on the traffic.
- **Analytics:** Here the various reading operations are represented as dashboards, reports, analytics and machine learning usages. Ad-hoc queries are allowed by Amazon Athena, and acceses to the raw S3 files is also allowed for the machine learning teams.

7 Conclusion

A cloud based Data Lake architecture was presented to assert all of the business and technical requirements, keeping in mind an horizontal scaled infrastructure, the distribution of points of failure, multiple data clients, data persistence and flexibility. The solution is based on Amazon Web Services, using a range of managed systems.

Next steps include data governance guidelines, and a project schedule for the implementation.

References

[1] *4 Guiding Principles for Modern Data Lake Architecture*. `https://www.upsolver.com/blog/four-principles-data-lake-architecture`. Accessed: 2022-06-05.

[2] *Cloud Storage as a data lake*. `https://cloud.google.com/architecture/build-a-data-lake-on-gcp`. Accessed: 2022-06-05.

[3] *Azure Data Lake*. `https://azure.microsoft.com/en-us/solutions/data-lake/`. Accessed: 2022-06-05.

[4] *Amazon Kinesis Firehose*. `https://aws.amazon.com/kinesis/data-firehose/`. Accessed: 2022-06-05.

[5] *Google Pub/Sub*. `https://support.google.com/cloud/answer/6255053?hl=en`. Accessed: 2022-06-05.

[6] *Apache Nifi*. https://nifi.apache.org/. Accessed: 2022-06-05.

[7] *Apache Kafka*. https://kafka.apache.org/. Accessed: 2022-06-05.

[8] *Apache Sqoop*. https://sqoop.apache.org/. Accessed: 2022-06-05.

[9] *Amazon S3*. https://aws.amazon.com/s3/. Accessed: 2022-06-05.

[10] *Amazon Athena*. https://aws.amazon.com/athena. Accessed: 2022-06-05.

[11] *Amazon Glue*. https://aws.amazon.com/glue/. Accessed: 2022-06-05.

[12] *Amazon Glue Data Catalog*. https://aws.amazon.com/blogs/big-data/metadata-classification-lineage-and-discovery-using-apache-atlas-on-amazon-emr/. Accessed: 2022-06-05.

[13] *Hadoop Distributed File System*. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. Accessed: 2022-06-05.

[14] *Amazon DynamoDB*. https://aws.amazon.com/dynamodb/. Accessed: 2022-06-05.

[15] *Amazon Elastic Computing Cloud*. https://aws.amazon.com/emr/. Accessed: 2022-06-05.

[16] *Apache Hadoop*. https://hadoop.apache.org/. Accessed: 2022-06-05.

[17] *Apache Pig*. https://pig.apache.org/. Accessed: 2022-06-05.

[18] *Apache Spark*. https://spark.apache.org/. Accessed: 2022-06-05.

[19] *Apache Airflow*. https://airflow.apache.org/. Accessed: 2022-06-05.