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
23/01/2023	Mohit Bansal	1.0	Initial draft

Reviewers / Approval

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

Key Contacts

Name	Role	Team	email
Mohit Bansal	Data Architect	Medical Data Processing	Mohit.bansal@payu.in

1. Purpose

Summary

Medical Data Processing Company's current data architecture can't keep up with the growth. As the volume of data continues to grow, the existing single node SQL Server is not able to scale. The SQL Server has become a single point of failure, hosting critical customer data. The CTO is looking for a data lake solution. This is the comprehensive design document with the proposed data lake architecture.

Document Contain

- Data Lake Requirements
- Data Lake Architecture design principles
- Assumptions
- Data Lake Architecture Proposal
- Design Considerations and Rationale
- Conclusion

Target Audience

- Company Leadership
- Data Engineer
- Data Scientist
- Product Team

In Scope and Out of Scope Items

- In Scope: The requirement, assumption, Architecture Diagram
- Out Scope: implementation of the data architecture, data governance, machine learning

2. Requirements:

Summary:

- Design system with high availability, fast and reliability
- Scale easily to keep up with the growth
- Break down data silos and maintain one source of truth
- Integrate flexibly with ML frameworks, reports and dashboards
- Provide fast querying system

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

Data Lake Architecture design principles

- Use event sourcing to ensure data traceability and consistency

  In a data lake architecture where compute and storage are separated, event sourcing should be used, and an immutable log of all incoming events should be maintained on object storage. Event sourcing enables you to retrace the steps to learn about the exact transformation applied on the raw data, down to the event level. If there was an issue in your ETL code, you can easily fix it and run the new code on the immutable original data.

- Keep it simple:

  The more complex data lake is, the more difficult it will be to use. Keep it as simple as possible so that everyone on your team can easily understand it.

- Choose the right storage technology:

  The type of storage technology will affect the performance of data lake. Choose the right one for your needs. There are several cloud storage technologies which can be used for data lakes. The most popular options are Amazon Simple Storage Service (S3), Azure Blob Storage, and Google Cloud Storage.S3 is a popular option because it offers a lot of flexibility and scalability.

- Keep the architecture open

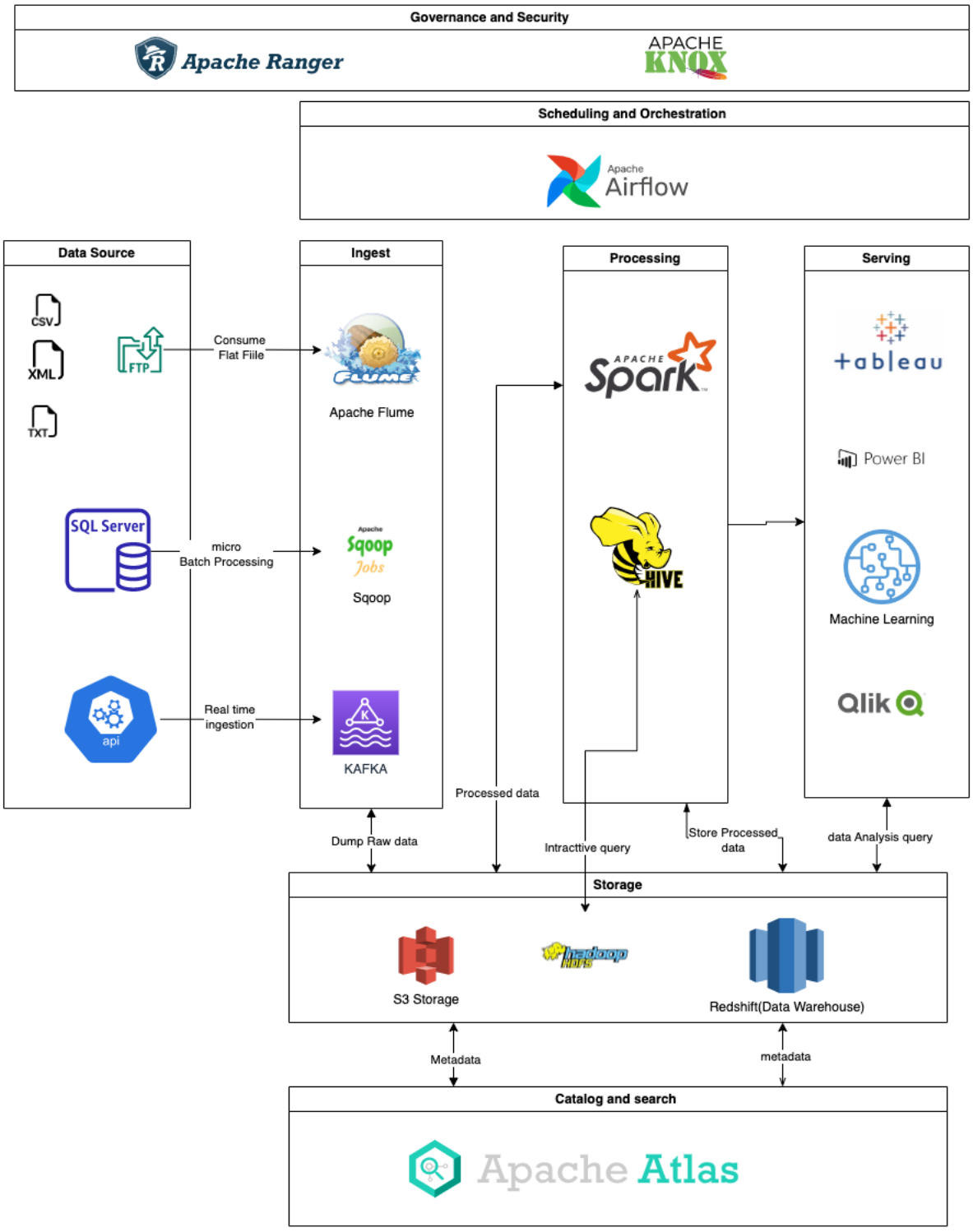
  To create an open architecture, you should:

  - Store the data in open formats like Avro and Parquet which are standard, well-known and accessible by different tools.
  - Retain historical data in object storage like Amazon S3.
  - Use a central meta-data repository such as AWS Glue. This will allow you to centralize and manage all your meta-data in a single location, reducing operational costs in infrastructure, IT resources and engineering hours.

3. Assumptions

- All data will be migrated to Cloud infra
- Raw data and clean data will be stored in separate storage like different s3 bucket
- Every dataset will have own data dictionary
- 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 more users or increasing data volume.
- For processing streaming or batch data will be processed on Spark cluster for fast and distributed manner
- Hurry to integrate all data in one physical environment without defining right processes around what the data means, what data already exists, and who owns the data. Thus, we end up creating multiple copies and no single source of truth for data.

4. Data Lake Architecture for Medical Data Processing Company



5. Design Considerations and Rationale

a. Ingestion Layer

AWS has several tools that can help to ingest data from different data sources eg; Direct connect, kinesis, snowball etc. But there is lot of open sources tools available for ingestion.
We have 3 different type of data source in our requirement like MySQL database, FTP servers where different file type is available txt, xml, csv and real-time API data.
We will use opensource tools that can ingest data to S3 and Redshift.

- **Apache Sqoop**: 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.
  In our requirement we will read history data from MySQL database server at nighttime and ingest into S3 Bucket to store data into database level.
- **Apache Flume:** Apache Flume is a tool 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 a highly reliable, distributed, and configurable tool. It is principally designed to copy streaming data (log data) from various web servers to HDFS.
  Apache Flume architecture consists of three major components, sources, channels, and sinks. A Flume source consumes events delivered to it by an external source like a web server. The external source sends events to the flume in a format that is recognized by the target Flume source. When a flume source receives an event, it stores it into one or more channels, which keep the event until it is consumed by a flume sink.
  In our requirement we will read file from FTP server and Flume will read and stream into S3 storage
- **Apache Kafka:** Apache Kafka is a distributed publish-subscribe messaging system and a robust queue that can handle a high volume of data and enables you to pass messages from one endpoint to another.
  In our requirement we will read transition log from API server and send to Kafka queue and read by s3 connect or spark streaming.

We can scale this open-source tool by deploying in auto scaling machine.
I consider AWS service like Direct connect, kinesis, snowball but they are costly and we can use open source library for this.

b. Storage Layer

This data lake architecture is based on Amazon S3 as the primary persistent data store. S3 storage can be used as compute and data processing storage. S3 storage provide on Demand scaling base storage requirement.

Aws provide S3 Cross-Region Replication so we can access storage in multi-region. It is also highly durable and low-cost with several options to connect.

Amazon S3 is secure by default. Amazon S3 supports user authentication to control access to data.

We can use Amazon S3 with Athena, Amazon Redshift Spectrum, and AWS Glue to query and process data. Amazon S3 also integrates with AWS Lambda serverless computing to run code without provisioning or managing servers.

Amazon S3 doesn't have any limits for the number of connections made to the bucket.

Data will be stored in the native formats upon ingestion. After ETL, data can be stored in open formats such as Apache Parquet, Avaro, and ORC that are standard, well-known and accessible by different tools.

The metadata can be populated by AWS Glue, the web interface or via the API, and stored in Apache Atlas. Apache Atlas is the one-stop solution for data governance and metadata management

Data can be shared with multiple data lakes through the S3 buckets. Data can be replicated in different regions for back-up and recovery.

Because SQL-based access is needed, Redshift is added to the storage layer. Redshift is a fast and fully managed petabyte-scale data warehouse that costs less than $1,000 per terabyte per year. The Redshift cluster can be resized to change the node type, number of nodes, or both.

I considered the cloud storage from Microsoft and Google. But Amazon S3 is a better choice in terms of the cost and performance.

c. Processing Layer

In Our requirement processing layer can have multiple solution for this. But We will choose opensource tools because they are available free of cost and easy to manage We considered Apache MapReduce, Hive and Spark as the processing.

- **Apache Hive**: Apache Hive is a distributed, fault-tolerant data warehouse system that enables analytics at a massive scale. A data warehouse provides a central store of information that can easily be analyzed to make informed, data driven decisions. Hive allows users to read, write, and manage petabytes of data using SQL. Hive gives a SQL-like interface to query data stored in various databases and file systems that integrate with Hadoop. Hive comes with command line tool and JDBC driver to connect the users to Hive programmatically. Hive provides necessary SQL abstraction to integrate SQL like queries, also known as Hive query language into the underlying Java without the need of implementing the queries in a low-level Java MapReduce API. Basically, to interact data stored on Hadoop, you do not need to write long and complicated MapReduce code in Java. You can interact with the same data by writing SQL queries and Apache Hive framework will convert the SQL queries into MapReduce code behind the scenes.
- **Apache Spark**: Apache Spark is a super-fast cluster computing technology designed for high-performance computation. It is based on Hadoop MapReduce and extends the MapReduce model to allow for more efficient use of it for different types of computations, such as interactive queries and stream processing. Spark's main feature is in-memory cluster computing, which increases an application's processing speed.
  Spark is intended to handle a variety of workloads, including batch applications, iterative algorithms, interactive queries, and streaming. It not only supports all these workloads in a single system, but it also reduces the management burden of maintaining separate tools.

Apache Spark supports batch and streaming processing. It can process real-time data by consuming a Kafka topic and storing it in an open format like parquet on S3. Spark can also be used to process massive amounts of data in batch for machine learning model training.

Apache Hive allows for ad-hoc queries on data stored in hdfs storage, which can be extended to S3 storage, as well as a SQL-like interface to query data stored in various databases.

I considered Apache MapReduce, Pig, Athena as the processing tool, but decided to use Apache Spark and Hive via EMR.

d. Serving Layer

The core task of the serving layer is to expose the views created by the process & analyze layer for querying by other systems or users. The serving layer consists of data that can be readily served to consumer applications. Hence this is mostly the processed data. serving layer is a method of presenting data to users.
Amazon QuickSight for business intelligence; and Amazon Elasticsearch Service for logs and text.
In our case we are using Tableau, Power BI and Machine learning tools for serving layer. From Processed layer we represent data on tableau dashboard which will refresh in real-time data which will use to get insight of data for another team.
Artificial intelligence (AI) and machine learning are becoming increasingly popular tools for building smart applications such as predictive analytics and deep learning. To make it more accessible, We can deploy ML platform which will consume processed data and serve to real-time incoming request via trained model.

e. Govern & Secure

Governance is key in building a managed data lake.
Security and Governance are important aspects of any Hadoop cluster. Security can be a challenge because of so many different services running in the same cluster.

- **Apache Knox:** The Apache Knox Gateway is an Application Gateway for interacting with the REST APIs and UIs of Apache Hadoop deployments. The Knox Gateway provides a single access point for all REST and HTTP interactions with Apache Hadoop clusters. Knox provides the following:
    - Proxying Services
    - Authentication Services
- **Apache Ranger:** Apache Ranger allows Centralized security administration for the Hadoop cluster.
  Ranger offers the following:
    - Manage all security-related tasks in a central UI or using REST APIs
    - Fine-grained authorization
    - Standardization of authorization method across all Hadoop components
    - Ranger supports different authorization methods - Role-based access control and attribute-based access control

8. Conclusion

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

- A data lake on AWS leverages S3 for secure, cost-effective, durable, and scalable storage.
- Redshift is added to the storage layer for SQL-based access and costs less than $1,000 per terabyte per year.
- Data can be quickly and easily ingested into Amazon S3 from the SQL Server, FTP server and APIs by Apache Sqoop, Apache Flume and Apache Kafka.
- For processing and analyzing the data stored in Amazon S3, AWS provides fast access to flexible and low-cost services, like Amazon EMR. On EMR we can use open-source tools like Apache spark and Apache Hive, and it is scalable.
- For Data Governance We will use Apache Atlas to storing meta data information of data which ingested into Hadoop system.
- For Security case we will use Apache Knox and Apache ranger to be given access based on requirement to user
- For Operations, Administration, monitoring we will use Apache Airflow
- For Serving the insight we will use Power BI, Tableau and for machine learning deploy ml platform.

9. References

1. Apache Sqoop (https://www.tutorialspoint.com/sqoop/index.htm)
2. Apache Flume (https://www.tutorialspoint.com/apache_flume/apache_flume_quick_guide.htm)
3. AWS S3 (https://docs.aws.amazon.com/whitepapers/latest/building-data-lakes/amazon-s3-data-lake-storage-platform.html)
4. Apache Atlas (https://atlas.apache.org)
5. Apache Hive (https://hive.apache.org)
6. Apache Spark (https://www.tutorialspoint.com/apache_spark/apache_spark_introduction.htm)
7. Apache Knox (https://knox.apache.org/)
8. Apache Ranger(https://ranger.apache.org/)
9. Apache Airflow(https://airflow.apache.org/)