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 |
| 10/25/2022 | Le Sy Khanh Duy | 1.0 | Initial draft |
| 10/29/2022 | Le Sy Khanh Duy | 2.0 | Change to open source |

## 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 | student@email.com |

# Purpose

Because of current data architecture of the company cannot scale up for the growth, and its single point of failure impacts to the business, we are looking for data lake solution to fix these issues.

This is the design document for that data lake architecture.

Document Outline:

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

Target Audience:

* Data lake stakeholders including:
* Company leadership
* Data engineers
* Data scientists
* External partners

In Scope and Out of Scope Items

* In scope: data architecture’s design.
* Out of scope: implementation, machine learning.

# Requirements

Summary of requirements for Data Lake:

# 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

* **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 can easily understand it.
* **Store all data in one place**: we should store all data in one place so that we can easily access it and analyze it. Additionally, storing all data in one place makes it easier to keep track of and comply with regulations.
* **Make sure it’s scalable**: data lake needs to be able to scale up or down as needed, depending on the amount of data we’re storing and processing. A data lake needs to be scalable so that it can easily handle the increasing volume of data being generated by businesses today.
* **Make sure it’s secured**: One important consideration when designing data lake is security. We need to make sure that your data is safe and secure, and that only authorized users can access it. There are several ways to secure our data lake, including data encryption, role-based access control and data masking.
* **Choose the right storage technology**: The type of storage technology we choose will affect the performance of our data lake. 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.
* **Make sure the data is organized and easy to access**:
  + This will make it easier to find what we’re looking for and reduce the time it takes to find and analyze the data.
  + Store data in standard format such as parquet, feather, Avro, so difference tools can access.
  + Use tool such as AWS Glue to manage meta-data, so difference tools can query our data (csv, tsv, parquet …)
* **Be prepared for changes**: The big data landscape is always changing, so be prepared for changes when designing our data lake. This will help ensure that our data lake can keep up with these changes and continue to provide value.
* **Keep focus on performance**:
  + Keep file size balance for read and write (split big file or merge small files for best ETL/query performance)
  + Think about how we use/query/analyze data, so we can partition data effective. (ex: if we query by time, we should partition our data by YYYYMMDD)

**Assumptions**

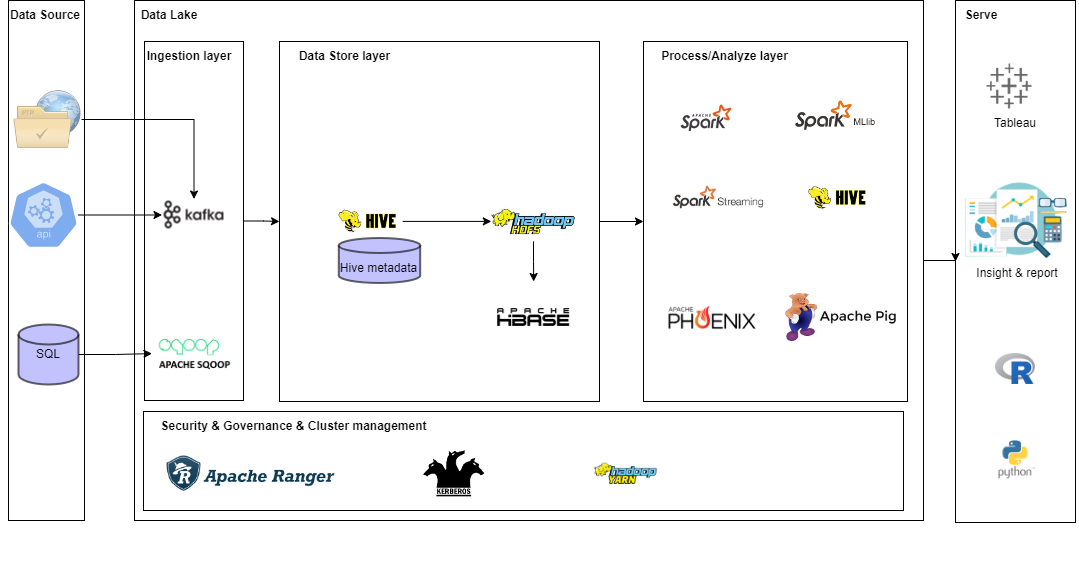
* + Put data to data lake as soon as possible. (real-time processing)
  + Use open-source services and on-premises storage to avoid vender lock-in
  + 20% data growth per year.
  + Current web applications must be refactoring to use new data architecture.

**Future risks:**

* + If we don’t manage well or lack of policies data lake will become Data swamp (unmanaged data lake that provide little value)
  + Cost to maintenance the data lake is bigger than cloud managed services if we use on-cloud services and storages.
  + Disaster.

# Data Lake Architecture for Medical Data Processing Company

* **Data Store layer**: store data without limit. (use cluster with n nodes to avoid single point of failure and able to store big data with no limit)
* **Ingestion layer**: load data from many data source and any format. ( use cluster with n nodes to avoid single point of failure )
* **Process/Analyze layer**: process/query/analyze big volume data from files or NoSQL database with SQL-like syntax.
* **Serve**: Serving the processed data to consumer applications.
* **Security & Governance & Cluster management**: secure for all Hadoop services and data. Manage metadata for other services access easier.



# Design Considerations and Rationale

## Ingestion Layer

Decide to use open-source services instead of cloud services to ingest data to data lake:

* Sqoop is a tool designed for effectively transferring bulk data between Apache Hadoop and structured data such as relational database. We choose Sqoop because it can work with many databases easy (MySQL, PostgreSQL, Oracle…)
* Kafka is used to build real-time streaming data pipelines and real-time streaming applications. We choose Kafka because it is distributed, fault tolerant. It can scale to 100s brokers (horizontal scalability).

I also considered cloud services or Flume, but I don’t because I want to keep it simple and don’t want to depend on a vendor:

* Direct connect establishes private connection between AWS and our data center. It increases bandwidth throughput – working with large data sets – lower cost. More consistent network experience.
* Snowball: high-secure, portable devices to collect and process data at the edge and migrate data into and out AWS. With snowball we can migrate petabytes data in short time.
* Kinesis Firehose: fully managed service, no administration, near real time, load data into S3. It is automatically scaling. Support many data formats.
* Kinesis Data Streams: Going to write custom code (producer / consumer), use with lambda to insert data in real-time to S3.
* Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. It has a simple and flexible architecture based on streaming data flows. Flume, just like Scoop Leverages Hadoop.

## Storage Layer

On-premises. We will use Hadoop distributed file system (HDFS) and HBase and Hive to store our data.

**Metadata**:

When put data to data lake, we must put them through Hive to manage metadata (or data catalog), these metadata will be stored in Hive metadata for other services can access and query data easier.

We should partition stored data by key that we use most when query (ex: datetime )

Metadata will include column name, data type, partition information…

**File format:**

Data format that stored by Hive should be Parquet, this is a standard format that can be used by many services/ tools.

**NoSQL database**:

We use HBase to store data that serve Web applications. HBase store data on HDFS cluster (n nodes), so we don’t worry about data growth.

**Avoid Single point of failure:**

We run 2 redundant Name nodes alongside one another, so that if one of the Name nodes fails, the cluster will quickly failover to the other Name node.

**Scale up ( plan for handling 20% data growth per year)**:

HDFS was designed as a scalable distributed file system to support thousands of nodes within a single cluster. With enough hardware, scaling to over 100 petabytes of raw storage capacity in one cluster can be easily and quickly

**Security:**

About security, we use HDFS Encryption to implements transparent end-to-end encryption, it will encrypt and decrypt without having to change application code.

We use Kerberos for authentication, validating whether a user exist and check his login credentials.

We also use Apache Ranger to enable authorization for our services in Hadoop.

(Validate to what data / services a user has access to)

**Backup and recovery data:**

Use HBase Live Cluster Backup tools to export data to HDFS.

Use HDFS snapshot to backup files.

Write custom script to backup Hive metadata.

**I consider** using Amazon S3 to store persistent data. But want to keep architecture pure open sources and avoid making architecture complicated. (Too many tools and techniques). I also worry about latency to and from system we integrate with.

S3 is an object storage to store and retrieve any amount of data. Cost effective with 99.999999999% of durability.

S3 also help us manage object life cycle easily.

For backup and restore, we will use S3 Glacier, it is a backup and long-term archival(multiyear) with extreme low cost.

To make data discoverable and usable, I use Glue Data Catalog and Glue Crawler to crawl data structure (column name, data type…) of files from S3 and manage them as metadata (we can also save metadata to DynamoDB)

We use Redshift for SQL-based access, we use Redshift as data warehouse for analytic purposes. With metadata crawled by Glue, we can move data from S3 to Redshift with simple ETL script/tools.

Redshift cluster can resize number of nodes, so we can scale up when data volume growth.

To secure our data, we use Amazon IAM to restrict access/read/write to specify data (resource-based policy) or base on user permission (user-based policy

## Processing/Analyze Layer

We use several open-source services for processing and analyzing:

* **The Apache Hive**: data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage using SQL. Structure can be projected onto data already in storage. A command line tool and JDBC driver are provided to connect users to Hive.
* **Apache Pig** is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets.
* **Apache Spark**is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters. It is faster MapReduce about 10 times because it use in-memory processing.
* **Apache Phoenix:** is a relational database layer on top of HBase, enabling SQL over HBase.

I also considered cloud services, but I don’t want to be locked-in with a vendor:

* **AWS Glue**: is a service that helps us making ETL job easy and fast to transform data from S3 to consumer. Glue also helps us crawl meta data and save it to Glue Data catalog, Analytic tools such as Athena can access to S3 based on these data catalog.
* **AWS Lambda**: use to write custom code for ETL.
* **Athena**: SQL liked query that access direct to S3. Athena is serverless, can be used as ad-hoc query tool.
* **EMR**: distributed compute framework (managed Hadoop environment), support for tools like Spark, Hive, HBase. With EMR we can process data on S3 at scale and on demand.
* **Machine Learning**: provides tools to create machine learning models and executing them on big data.
* **QuickSight**: BI service that can access data from Redshift.

## Serve Layer

* The main task of the serving layer is to expose the result created by the process & analyze layer for querying by other systems or user or application.
* data served by process/analyze layer can be stored in HBase database or parquet files based on access method of consumers.
* Result data may be done by a job trigger when data come to data lake or we can provide on-demand api/ function that consumers can call anytime they want.

# Conclusion:

To summarize, our data lake is on-premises HDFS storage,with open-source services and tools to ingest, process and analyze data . the output will be served as a group of files or an API to consumers that may be a business intelligence tools, web application .

The next step of this project are :

* List out all use cases.
* List out all consumers, what tools/method/protocol they use to access the data lake.
* List out all the impact to the existing applications, services that using data in the old database.
* After understand clearly all use cases, accessing methods , we must re-think about the tools/technology we should use to map with that use cases.

# References:

<https://hadoop.apache.org/>

<https://pig.apache.org/>

<https://hive.apache.org/>

<https://hbase.apache.org/>

<https://spark.apache.org/>

<https://kafka.apache.org/>

<https://sqoop.apache.org/>

<https://www.cohesity.com/blogs/why-your-hadoop-backup-recovery-strategy-is-falling-short/>

<https://svn.apache.org/repos/asf/hbase/hbase.apache.org/trunk/0.94/book/ops.backup.html>

<https://www.linkedin.com/pulse/hadoop-cloud-vs-premises-shahebaz-shaik>

<https://www.indiumsoftware.com/blog/data-lake-architecture-design/>

[https://aws.amazon.com/products/?aws-products-all.sort-by=item.additionalFields.productNameLowercase&aws-products-all.sort-order=asc&awsf.re%3AInvent=\*all&awsf.Free%20Tier%20Type=\*all&awsf.tech-category=\*all](https://aws.amazon.com/products/?aws-products-all.sort-by=item.additionalFields.productNameLowercase&aws-products-all.sort-order=asc&awsf.re%3AInvent=*all&awsf.Free%20Tier%20Type=*all&awsf.tech-category=*all)