

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 |
| 06/04/2020 | Chrysanthi Polyzoni | 0.2 | Initial draft |

## Reviewers / Approval

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

## Key Contacts

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Role | Team | email |
| FirstName LastName | Data Architect | Medical Data Processing | chpolyzo@gmail.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

## What is the purpose of the document (Summary)

This document is a walkthrough of all the steps and the tools used to create a Data Lake in the cloud. The document contains a description of an architecture proposed to implement an efficient and cost effective way to use data to solve the specific business needs raised after an emergency in the data center. This document contains an integrated strategy to implement a cost-effective way to store all data of an organization for later processing. Data Analysts within the company can focus on finding meaningful patterns in data with more visibility and easy access to the required data. This document has been created to describe technical steps taken to create a Data Lake in the cloud. Target audience of this artifact is a *highly technical group of people* including enterprise architects, software engineers, and technical directors.

### In scope

1. **Understand state of the art and challenges of existing architecture**
2. **Understand critical points of failure and how to avoid those in a future solution**
3. **Understand who uses data for decision making and who could potentially use it to add value**

### Out of scope

1. **Business model innovation and business expansion with the new architecture**
2. **Competition framework in which the company is operating and how Data lake will have a higher competitive ranking with the new architecture**
3. **Explore and exploit business models to discover different revenue streams because of the new architecture.**

# Requirements

## Summary of requirements for Data Lake. Summarize your understanding of the problem statement.

* 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

## 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

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

# Data Lake Architecture design principles <approx. ½ page>

**Design Principles. Baseline criteria to design the system and rules/guiding principles to be followed**

Break data silos

Create a unique pool of data so that all teams can have access to the same information

Have Finance and Operations working on the same page and create common KPIs

Enable innovation

Real time data analytics integration

Predict the unknown

Create a resilient culture where you are better prepared for the next disaster, and have to build the ability to learn and adapt without loss making.

Self-service access is immediate and does not take weeks or months for people to get the data they need.

Perform the ability to readily identify new revenue streams, recognize patterns, provide new types of previously unknown or unattainable information for decision making).

**Rationale of “why” the design principal was selected and how this can help in overall Data Lake design long-term for Medical Data Systems**

It is mandatory to create a resilient data system capable to respond to both technical and ethical needs of people trusting the company.

# Assumptions

We assume that files once written are only read, most of the times sequentially, mostly archival data. We assume most files are mutated by appending new data instead of overwriting existing data. The system must constantly monitor itself and detect, tolerate and recover promptly from component failures on a routine basis. The system stores a modest number of small files. Workloads have small sequential writes that append data to files. Once written, files are seldom modified again.

This is an initial problem statement. We assume we will be working together in order to provide a seamless data experience. One thing to consider is cloud provider we need to decide upon existing competition framework. Final tools to be used will be defined upon decision of the cloud provider.

**Potential Risks that may be created now or in future based on our assumptions**

1. Hadoop cluster will use Linux operating system
2. Unstructured data and lack of metadata can lead to a Data Lake becoming a Data "Swamp", where it is hard to find useful data.
3. Data scientists may require additional training to successfully mine data from a Data Lake.
4. Inexperienced users can start dumping data into a Data Lake without a viable strategy or plan to extract valuable insight.

# Data Lake Architecture for Medical Data Processing Company

Graphical user interface, diagram, application

Description automatically generated

# Design Considerations and Rationale <at least 3 pages>

## Ingestion Layer

* Apache Kafka for ingesting log data. Kafka provides a robust, reliable and highly scalable solution for processing and queueing streaming data, and is probably the most common building block used in streaming architectures.
* Apache Sqoop: ingesting relational data sets
* Apache Hudi: Ingest Change Data Capture (CDC) as a mechanism to snapshot, restore or rollback changes in data when troubleshooting resulting from poor data quality in upstream systems. In this way we also ensure smooth application of the California Consumer Privacy Act as well as travel time views, meaning it is now possible to see how data looked in a given point in time. Hudi supports fast and pluggable indexing for all data stored in a database storage, has the ability to automatically publish data along with the support of rolling back the operation just like you would do in your database. Hudi also offers the timeline view of the dataset by tracking the lineage and maintaining a timeline metadata. Last but not least Hudi automatically manages the file sizes, the compaction process and the layout using statistics.

Hudi datasets are written in two different ways. Ffirst using a tool called Delta Streamer, which can connect to sources like S3, apache Hive and apache Kafka. As soon as a file reaches the system, Delta Streamer captures the event to be produces and simplifies the overall process by applying all necessary updates into the Hudi Dataset. Secondly, Hudi datasets can also be written using Spark data sources allowing you to interact with your dataset within the existing Spark applications.

* Steaming data from a variety of sources in case this happens, can be ingested using a combination of Kafka and flume frameworks

We Keep a copy of the Database on HDFS, that is Hadoop Distributed File system, a managed Hadoop framework that runs on the cloud. Apache Sqoop is the tool to perform this bulk process. Once we have a copy of the database existing APIs or FTPs will be replaced by appropriate tools in distributed frameworks and Data lakes.

As described above, in order to ingest existing and new data to HDFS Kafka and Sqoop is proposed, while ingesting data from HDFS to Data lake, Delta Streamer and Spark Frameworks will be used to facilitate a functional and up to date comprehensive data source for effective and data driven decision making.

The ingestion layer design would scale through the Hadoop Distributed File System (HDFS) which is a distributed file system designed to run on commodity hardware. Hardware failure is the norm rather than the exception. An HDFS instance may consist of hundreds or thousands of server machines, each storing part of the file system’s data. The fact that there are a huge number of components and that each component has a non-trivial probability of failure means that some component of HDFS is always non-functional. Therefore, detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

**Other 3rd party tools, open source tools considered but did not make it to the architecture because it is preferable to start over with well-known tools like the ones proposed to complete troubleshooting fast and having a seamless transition from MySQL to distributed NoSQL systems all the way to a data Lake which doesn’t risk to become a data swamp.**

**There are many other shortcomings to the selection of tools like**

**Apache NiFi supports powerful and scalable directed graphs of data routing, transformation, and system mediation logic.**

**Amazon Kinesis makes it easy to collect, process, and analyze real-time, streaming data so you can get timely insights and react quickly to new information. Amazon Kinesis offers key capabilities to cost-effectively process streaming data at any scale, along with the flexibility to choose the tools that best suit the requirements of your application. With Amazon Kinesis, you can ingest real-time data such as video, audio, application logs, website clickstreams, and IoT telemetry data for machine learning, analytics, and other applications. Amazon Kinesis enables you to process and analyze data as it arrives and respond instantly instead of having to wait until all your data is collected before the processing can begin. However we are yet to decide the cloud provider.**

**Apache Flume: Kafka can support data streams for multiple applications, whereas Flume is specific for Hadoop and**[**big data analysis**](https://www.educba.com/big-data-analytics-techniques/)**. Kafka can process and monitor data in distributed systems whereas Flume gathers data from distributed systems to land data on a centralized data store.**

## Storage Layer

Applications that run on HDFS have large data sets. A typical file in HDFS is gigabytes to terabytes in size. Thus, HDFS is tuned to support large files. It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster. It should support tens of millions of files in a single instance.

Hadoop On Demand (HOD) is a system for provisioning virtual Hadoop clusters over a large physical cluster. It uses the Torque resource manager to do node allocation. On the allocated nodes, it can start Hadoop Map/Reduce and HDFS daemons. It automatically generates the appropriate configuration files (hadoop-site.xml) for the Hadoop daemons and client. HOD also has the capability to distribute Hadoop to the nodes in the virtual cluster that it allocates. In short, HOD makes it easy for administrators and users to quickly setup and use Hadoop. It is also a very useful tool for Hadoop developers and testers who need to share a physical cluster for testing their own Hadoop versions.

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

**Metadata information**

On HDFS the master node stores three major types of metadata: the file and chunk namespaces, the mapping from files to chunks and the locations of each chunk’s replicas. All metadata is kept in the master’s memory.

One advantage HDFS has over S3 is metadata performance: it is relatively fast to list thousands of files against HDFS name node but can take a long time for S3.

Apache Atlas installed on [Amazon EMR](https://aws.amazon.com/emr/) can provide capability for the use of metadata, cataloging, and data lineage. You can use the proposed setup to dynamically classify data and view the lineage of data as it moves through various processes. As part of this, you can use a domain-specific language (DSL) in Atlas to search the metadata. Apache Atlas is an enterprise-scale data governance and metadata framework for Hadoop. Atlas provides open metadata management and governance capabilities for organizations to build a catalog of their data assets. Atlas supports classification of data, including storage *lineage,*which depicts how data has evolved. It also provides features to search for key elements and their business definition.

Among all the features that Apache Atlas offers, the core feature of our interest in this post is the Apache Hive metadata management and data lineage. After you successfully set up Atlas, it uses a native tool to import Hive tables and analyze the data to present data lineage intuitively to the end users. To read more about Atlas and its features, see [the Atlas website](https://atlas.apache.org/).

**Atlas Features**

Metadata types & instances

* Pre-defined types for various Hadoop and non-Hadoop metadata
* Ability to define new types for the metadata to be managed
* Types can have primitive attributes, complex attributes, object references; can inherit from other types
* Instances of types, called entities, capture metadata object details and their relationships
* REST APIs to work with types and instances allow easier integration

Classification

* Ability to dynamically create classifications - like PII, EXPIRES\_ON, DATA\_QUALITY, SENSITIVE
* Classifications can include attributes - like expiry date attribute in EXPIRES\_ON classification
* Entities can be associated with multiple classifications, enabling easier discovery and security enforcement
* Propagation of classifications via lineage - automatically ensures that classifications follow the data as it goes through various processing

Lineage

* Intuitive UI to view lineage of data as it moves through various processes
* REST APIs to access and update lineage

Search/Discovery

* Intuitive UI to search entities by type, classification, attribute value or free-text
* Rich REST APIs to search by complex criteria
* SQL like query language to search entities - Domain Specific Language (DSL)

Security & Data Masking

* Fine grained security for metadata access, enabling controls on access to entity instances and operations like add/update/remove classifications
* Integration with Apache Ranger enables authorization/data-masking on data access based on classifications associated with entities in Apache Atlas. For example:
* who can access data classified as PII, SENSITIVE
* customer-service users can only see last 4 digits of columns classified as NATIONAL\_ID

**Format**

Parquet is an open source file format available to any project in the Hadoop ecosystem. Apache Parquet is designed for efficient as well as performant flat columnar storage format of data compared to row based files like CSV or TSV files.

More specifically Apache Hudi

*Copy on Write table*

1. Read out record from parquet files
2. Merge records according to passing update records
3. Write merged records to files
4. Commit to table commitActionExecut

*Merge on Read table*

1. Store data records into AVRO format log file
2. Scheduled compaction

*Indexing*

1. Mapping Hudi record key (in metadata column) to file group and file id
2. In-memory, bloom, filter and HBase

*Table level APIs*

1. UPSERT

Apache Ranger is a framework to enable, monitor and manage comprehensive data security across the Hadoop platform.

The vision with Ranger is to provide comprehensive security across the Apache Hadoop ecosystem. With the advent of Apache YARN, the Hadoop platform can now support a true data lake architecture. Data security within Hadoop needs to evolve to support multiple use cases for data access, while also providing a framework for central administration of security policies and monitoring of user access.

**Different 3rd party tools can be considered. In an on-going process where we still have to discuss more in detail data to be ingested and how would this grow other third party tools or open source tools can be considered. We have considered a solution based on AWS but if we believe this is not the solution for us there are plenty of alternatives such as Azure Data Lake Storage Gen 1 and Gen 2, Google cloud storage, IBM solutions. We have not considered those because AWS is better and more straightforward to understand in the drafting situation like the one we are stated in having plenty of documentation we can take inspiration from.**

## Processing Layer

The *Processing Layer* runs user queries and advanced analytical tools on structured data. Processes can be run in real-time, as a batch, or interactively. Business logic is applied in this layer and data is consumed by analytical applications. This layer is also known as *trusted, gold*, or *production-ready*. Data will be processed through Spark, Presto and Hive queries.

Change Data Capture refers to the process or technology for identifying and capturing changes made to a dataset. Those changes can then be applied to another data set or made available in a format consumable by data integration tools. This is typically done to keep systems in sync and to maintain data record history as it changes over time.

Delta Lake is an open-source storage layer that brings ACID transactions to Apache Spark and the big data workloads. So from its architecture, a picture of it if we could see that it has at least four of the capability we just mentioned. like support for both Streaming and Batch. We could consider using that one.

[Apache Hive](https://en.wikipedia.org/wiki/Apache_Hive) is a ‘big’ data warehouse framework that supports analysis of large datasets stored in Hadoop’s HDFS and compatible file systems such as Amazon S3, Azure Blob, and Azure Data Lake Store File systems. Over the years, Hive has proven to be great for batch ETL and is very robust at scale.

[Apache Spark](https://en.wikipedia.org/wiki/Apache_Spark) is an in-memory cluster computing framework originally developed at the University of California, Berkeley’s AMPLab. Spark excels in use cases like continuous applications that require streaming data to be processed, analyzed, and stored. Spark is also used for batch/streaming ETL and is very robust at scale, but using and practicing Spark requires a completely different skill set that is above and beyond SQL.

[Presto](https://en.wikipedia.org/wiki/Presto_(SQL_query_engine)) is an in-memory distributed SQL query engine for running interactive analytic queries. Developed out of Facebook, Presto plays a vital role in providing accelerated access to any data store and helps avoid the need to move activated/refined datasets to an on-premises or cloud MPP data warehouse for analytics and reporting.

Here below possibilities we can further discuss:

Spark is intended to enhance not replace, the Hadoop stack. From day one, Spark was designed to read and write data from and to HDFS, as well as other storage systems, such as HBase and Amazon’s S3. As such, Hadoop users can enrich their processing capabilities by combining Spark with Hadoop MapReduce, HBase, and other big data frameworks.

Apache Spark is the de facto open source big data processing engine, enabling SQL queries and rapid distributed processing of the data in your data lake.

The Databricks Unified Data Analytics Platform makes it easy to run SQL queries on your data lake, do massive scale data engineering and collaborative data science.

Alternatively to Hudi, it is possible to simplify and strengthen data architecture by using [Delta Lake](https://delta.io/) to ensure data validity and consistent views at petabyte scale. While Hudi is more focused on streaming data, delta lake could be yet another possibility to further evaluate.

Amazon Web Services’ Simple Storage Service (S3) provides cost effective object storage for data lakes. This is an alternative solution to HDFS but is not fault tolerant, this means it will not keep three copies of the files and therefore more risk prone.

Presto was originally created by Facebook to run queries on Hadoop data warehouses. It can be used to run SQL queries on data lakes at scale.

**Other 3rd party tools, open source tools have been considered, however we still need to consider alternative tools. The more we deepen our understanding on the company’s specific needs the more we will find the perfect solution to accomplish. Again this is a draft document describing an easy to understand situation where we can take inspiration from AWS solutions but we can use different technologies if we were to decide using only S3 over HDFS. One advantage HDFS has over S3 is metadata performance: it is relatively fast to list thousands of files against HDFS name node but can take a long time for S3. However, the**[**scalable partition handling feature**](https://databricks.com/blog/2016/12/15/scalable-partition-handling-for-cloud-native-architecture-in-apache-spark-2-1.html)**we implemented in Apache Spark 2.1 mitigates this issue with metadata performance in S3. Spark, Presto, Redshift, SnowFlake. The proposed architecture working mainly on HDFS can well scale in all these frameworks.**

## Serving Layer

The core task of the serving layer is to expose the views created by both the batch and speed layer for querying by other systems or users.

Final views and dashboards are planed to be stored here.

All users will be able to access the system with different roles and permissions.

# 8. Conclusion

Further discussions have to be taken in order to comprehensively create the successful solution for the company. We can ingest different kinds of data and we need to understand if we have to do with batch data or streaming data, or both, if we can connect streaming functionality of machines in different medical facilities, what is the current situation of data sharing, who owns data and how different permissions should be given in the name of privacy. Given disasters already taken place it is important to prevent rather than respond and create a solid bases to work on.

# 9. References

<https://medium.com/weareservian/faster-change-data-capture-for-your-data-lake-6ad9d743074c>

<https://www.qubole.com/blog/embrace-big-data-choice-curate-and-analyze-data-with-hive-spark-and-presto/>

<https://www.qubole.com/blog/embrace-big-data-choice-curate-and-analyze-data-with-hive-spark-and-presto/>

<https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Introduction>

<https://databricks.com/spark/about>

<https://databricks.com/blog/2014/01/21/spark-and-hadoop.html>

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

<https://aws.amazon.com/blogs/big-data/building-and-maintaining-an-amazon-s3-metadata-index-without-servers/>

<https://aws.amazon.com/kinesis/>

<https://aws.amazon.com/blogs/big-data/metadata-classification-lineage-and-discovery-using-apache-atlas-on-amazon-emr/>

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