Data Lake Architecture -

A Comprehensive Design Document

Medical Data Processing Company

Data Architect: Mateus Leao

# Tracker

## Revision, Sign off Sheet and Key Contacts

## Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Change Reference |
| 06/04/2020 | Mateus Leao | 0.1 | Initial draft |

## Reviewers / Approval

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

## Key Contacts

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Role | Team | email |
| FirstName LastName | Data Architect | Medical Data Processing | student@email.com |

# Purpose

The purpose of this document is to be able to delivery a technical and business-oriented document that exposes a data and a system problem and proposes a Data Lake as a solution to this problem.

This document contains:

* Requirements
* Data Lake Architecture Design Principles
* Assumptions
* The Diagram of the Architecture proposed for Medical Data Processing Company
* Design considerations about Data Lake Layers
  + Ingestion Layer
  + Storage Layer
  + Processing Layer
  + Serving Layer

The target of this document is anyone into the Medical Data Processing Company that is interested into this project. Overall, we aim to executives to read this document and support and push the proposal to happen.

This document has the proposal and the architecture details and requirements, but it doesn’t have considerations about the implementation part of it, which would happen to be in a separate document and effort.

# Requirements <approx. 1 page>

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

<Existing Technical Environment>

<Current Data Volume>

<Business Requirements>

<Technical Requirements>

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

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

<List of the design principles. What is the baseline criteria to design the system? What rules/guiding principles should be followed?>

Our architecture was planned to work with four layers: Ingestion Layer, where we plan to have a Kafka Cluster that’s gonna persist the data for 7 days, and that will be sending Data to our storage (S3 or HDFS) with the help of Apache Hudi.

In the Storage Layer we’ll have HDFS or S3, we have a staging area that is going to receive the raw data from Kafka. Storage Layer and Processing Layer work together, and is only with the help of Spark that we can write data from our Staging Area to a more consolidated data storage area, in this area we’ll have our Derived Tables, these tables have the Data in a format that works for the Business and for Analytics Serving according to our query patterns and needs.

The Data will be available in .parquet and .avro formats.

<Provide rationale of “why” the design principal was selected and how would it help in overall Data Lake design long-term for Medical Data Systems>

[You may not use this example in your final solution]

e.g.: Leverage open source tools as much as possible

# Assumptions <approx. ⅓ page>

<What are the assumptions you have made while creating the Data Lake architecture?>

<Be creative, what questions did you have while designing the architecture?>

<What data is missing in the problem statement, and you made assumptions about it to create the architecture?>

**Assumptions:**

HDFS would perform better than S3, but will be more costly. S3 is a viable option.

The Apache Cluster will work without a single point of failure.

Linux O.S is our best option for the Spark/HDFS Cluster.

The AWS Kafka Cluster is more performant than a self-managed cluster.

**Risks:**

If we use S3 for the storage it could be that the performance will be too slow to satisfy our real-time apps needs.

If Kafka has a single point of failure then we risk to lose the data that is being sent to Kafka.

<Describes any potential risks that may be created now or in future based on these assumptions>

[You may not use this example in your final solution] e.g.:

1. Hadoop cluster will use Linux operating system
2. Data Lake will not support X, Y, Z

# Data Lake Architecture for Medical Data Processing Company

Diagram

Description automatically generated

# Design Considerations and Rationale <at least 3 pages>

## Ingestion Layer

We will have applications and databases working with the Kafka Connect API. We will use Kafka Sources so we can ingest the data coming from most of our Facilities and equipment into **Kafka Topics.** This is valid for data coming from databases and APIS. For the data coming from the FTP Servers we´ll be sending this data directly to our HDFS Storage, with the help of Apache Hudi + Apache Spark.

The main tools in our architecture are:

**Apache Kafka:**

Works as a distributed queuing system capable of handling huges amounts of data for both actions of receiving data (sources) or sending data (sinks)

**Apache Hudi**:

Hudi helps us manage the metadata of our tables. Apache hudi works with Apache Kafka and gives us a way to incrementally add data to our tables.

**HDFS or S3 Storage:**

The core of our data lake, where our data will be stored. HDFS and S3 are scalable distributed storage systems where our data will live in. We will have different environments, one where data will be stored as it comes (raw), and another where data will be stored after being processed, cleansed, or aggregated with the help of Apache Spark.

**Apache Spark:**

Another core part of our architecture. Spark is responsible for the processing of data within our Data Lake. Scalable can work with the same processing cluster as HDFS works (EMR, by AWS, for example). Spark is a distributed processing tool capable of handling huges amount of data in a very efficient manner. Spark is responsible for sending data from staging to our derived data environment, with Spark it´s possible to do many types of transformations: Joins, Deduplication, Aggregations, Filter data, etc.

**Scalability Considerations:**

Due to the distributed nature of the applications we are going to use, this architecture will be able to easily scale so it can handle larger and larges amount of data. If we are using AWS EMR we can increase our cluster machines (vertically), or increase our cluster horizontally (more machines). The same is valid for the Kafka Cluster.

We should have monitoring and usage metrics in place so we can receive alerts when it´s time to increase the size of our clusters.

**Considerations about HDFS vs S3:**

There is the possibility in this Architecture of using HDFS (Hadoop File System) or S3 for the storage of our data. HDFS is able to handle better higher throughputs of Data in comparison to s3. On the other hand, S3 can be much cheaper than HDFS, additional tests would have to be done to make sure that S3 can handle the workload if that´s the decided way to continue.

<How do you plan to ingest different types of data?>

<How would you ingest data coming from Databases, FTP servers, APIs?>

<What tools would be used? Why? >

<How would the ingestion layer design scale?>

<What other tools were considered? (3rd party tools, open source tools considered but did not make it to the architecture you are proposing). Are there other shortcomings to your selection of tools? If so what? Does the 3rd party tool solve that?>

## Storage Layer

**Using HDFS:**

HDFS stands for Hadoop File System. With HDFS Data is stored in file formats (CSV, Parquet, Avro, JSON, etc). HDFS integrates well with other systems from our Architecture.

HDFS will receive data from Apache Kafka, from the FTP, and also from Apache Spark Jobs. All data will be stored in a file system in a Hadoop cluster that can be scaled both in Disk as in Processing or Memory.

**How would HDFS handle 20% YoY Data Growth?**

The disk of the cluster has a fixed size, every working node can have a disk associated to it, the more nodes we have the more disks we can have. These disks form a pool of resources that HDFS can manage. In case the disks are getting filled then it´s possible to simply increase the disk size of every node, or increase the number of nodes and disks.

**OR using S3:**

S3 would work similarly to HDFS, it integrates well with Kafka, Apache Hudi, and Spark.

S3 is much cheaper than HDFS. Besides that, with S3 we can decouple our Spark Processing from our Storage, which is something nice to have.

**How would S3 handle 20% YoY Data Growth?**

S3 is a service fully managed by AWS that increases infinitely (almost). No configurations have to be done, getting more points to S3. No one would have to worry about scaling the size of the S3 buckets because they’re unlimited.

**Backup and Recovery Considerations and Strategy:**

For recoveries we can count with the data that Kafka sends to MongoDB, so if something fails when sending data from Kafka to our staging area, then we could just reprocess and get the data that got errored out. But if something more serious happen and we lose the data from our staging area in HDFS or S3, then we could go to Mongo, that would be persisting our Data.

If, let´s say, a Spark Job fails when sending data from Staging to our Derived tables, there are two things we could do, one is allowing our pipeline to retry the job, so if the problem is intermittent or just a failure in the Hadoop/Kafka Cluster, retrying could solve or problem.

Data is being persisted in Mongo as a Backup Source. The Mongo Cluster won’t be accessible externally, and will be a single source of truth for our Data, allowing data recoveries to be made through the API Gateway that will connect to Mongo.

**Considerations about Metadata:**

**Data Formats:**

The data in our Warehouse will be in the Parquet and in the Avro Format. For historical querying the data will be in Parquet format only, and for incremental querying the data will be available in the Avro format. Apache Spark can work with both file types.

**Considerations about data security:**

<How do you plan to store a vast amount of data? >

<How would the system handle 20% YoY Data Growth rate?>

<How do you plan to handle back-up and recovery? What are the strategies?>

<How do you plan to store custom **metadata** information? What type of information would metadata hold?>

<What format of the data do you plan to use? Why?>

<How do you plan to secure data (at a high-level)? Identify 2-3 techniques/tools/considerations>

<What other tools were considered? (3rd party tools, open source tools considered but did not make it to the architecture you are proposing). Are there other shortcomings to your selection of tools? If so what? Does the 3rd party tool solve that?>

## Processing Layer

The processing Layer is responsible for working with the data coming from our staging environment. Spark is our main player here; it works as a central processing engine capable of processing and doing multiple types of transformations with our Data. After doing the needed transformations, merges, joins, and others, the Data is stored in our Derived Tables, the Derived tables can have aggregated data, historical data, or simply data that has been processed and is in a more analytics-ready format.

The processing layer applies business rules to the Data, together with Quality operations that give more value to our Data.

**Batch vs Realtime vs CDC:**

Spark and Airflow are responsible for the Batch processing of our Data, with Airflow we can have CRON trigger that run spark jobs to process and generate data according to our needs.

Realtime Data can be queried from our derived tables, when data gets into staging we have streaming spark applications that transform the data and store it in our Derived environment.

CDC: Apache Hudi is responsible for managing the changes of Data, we are able to upsert data and handle incremental querying and data processing.

Enabling Ad-hoc queries:

**Scalability Considerations:**

We can use auto-scaling configurations within our EMP AWS Cluster, allowing us to easily handle jobs that require more processing power. After configuring auto-scaling in the Spark Cluster there´s nothing more to be done, AWS handles everything. We can scale our cluster both vertically and horizontally.

We should have a metric that shows our Memory usage since Apache Spark uses memory to cache our Data.

**Ad-hoc queries support:**

**Different tools involved in the Processing Layer:**

## Serving Layer

What is the serving layer?

Types of Data Available in the serving layer:

Uses of the Serving Layer in respect to Data:

Considerations about AWS Athena:

# 8. Conclusion

This project was made to provide a robust architecture that is able to ingest and process and make available huges amounts of data in real time (PB Scale) with a very high throughput, additionally to that we have proposed systems that are able to scale up to petabytes of data, making this architecture sustainable in the long run.

# 9. References

<https://www.xenonstack.com/insights/what-is-hudi/>

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