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 |
| 04/10/2021 | Eric Lok | 0.1 | Initial draft |
| 04/11/2021 | Eric Lok | 0.2 | Added assumptions |
| 04/12/2021 | Eric Lok | 0.3 | Added rationale and conclusion |

## Reviewers / Approval

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

## Key Contacts

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Role | Team | email |
| Eric Lok | Data Architect | Medical Data Processing | Eric.lok@gmail.com |

# Purpose

This document will go through the business needs of a company called Medical Data Processing. We will lay out the processes, inputs, and outputs. We will walk through the challenges they face and propose strategy to their infrastructure that will solve their current problems.

This document is targeted towards a technical audience. The assumption is that this audience already understands the issues with the current infrastructure and how that process is currently put together.

The goal of this project is to redesign and improve the data layer – the ingesting, processing, storage, and serving of data. It does not necessarily aim to improve reporting, analytics, or the web application layer. As we hope to increase the stability of the data, we also indirectly increase the stability of the web applications and reporting to the users, however, we will not change the functionality itself.

# Requirements

Medical Data Processing has had significant growth since 2007 with about 1100 customers and 370 employees. Their solution is used by about 8000 individual medical care facilities. Their current architecture consists of 3 layers – web server, application, and data. The data layer is backed by SQL Server.

The input data comes from over 8000 individual facilities with different data formats such as XML, TXT, and CSV. The mechanisms used to retrieve this data are calling a customer API or FTP. It can be a customer’s FTP in which company is *pulling* data or company is hosting an FTP in which customers are *pushing* data to. All the files are compressed with gzip format and password protected.

Over the years, company has accumulated over 8TB of data running on a single node SQL Server. Without the ability to scale, this single node can only process the data nightly due to the compute capacity limitations. In addition, ETL processes and SQL reporting queries are running slow.

**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

The design of the proposed data lake architecture is based on a set of principles:

**High Availability**: a proper system should have high availability, which means the service should be available more than 95% of the time. In order to accomplish this, we need to establish a fail-over plan. We aimed for the system to be reliable and fault tolerant.

**High Performance**: An important requirement for Medical Data Company is real time analytics. In order to support real time data, our infrastructure must offer high performance. Specifically, we must provide the ability to scale horizontally instead of vertically. Scaling vertically means adding CPU and ram to a computer and you can only go so far. Scaling horizontally means we can add more common hardware to a cluster of common hardware to increase performance.

**Scalability**: In order to handle increased data, volume, or clients, we must be able to scale along with the business needs. Going hand in hand with high performance, the ability to scale horizontally gives us the ability to scale dynamically with business needs.

**Microservice**: Historically we’ve built monolithic applications – one large infrastructure to handle all requirements. We have tried to build a one-size-fits-all solution. As our problem size becomes larger, these types of solutions no longer work. A large problem must be broken down into smaller problems and those smaller problems must be handled individually by a respective solution. For example, instead of using one software to handle ingestion, processing, and storage, we break that problem into three smaller problems and find a solution for each one and handles each one at its best.

# Assumptions

The following are assumptions about Medical Data Company, its employees, and infrastructures. If these assumptions are incorrect, we must address them with the necessary stakeholders.

**Operating system**: It has not been stated what operating system the company uses. Apache software generally runs best on Linux, however, if the company does not have the staff to support Linux, we might have to run the infrastructure on Windows.

**Cloud vs On-Premise**: Infrastructure can now easily run on cloud services such as Azure, AWS, and Google Cloud. This proposal does not address whether the applications will run on the cloud, on-premise, or hybrid.

Datacenter location for speed/performance

**Data Structure**: – Although we’ve address the format of the data (e.g. CSV vs JSON, etc.), we have not addressed the structure of the data. It is unclear whether the we are dealing with structured vs unstructured data. However, the proposed solution should be able to handle both.

**Programming Language**: Spark is a framework supported by many programming languages. However, it is not clear what language the company is experienced with. The assumption is that Spark will support the language they currently use. Potential risk is Spark does not support a programming language that the company uses.

**Budget**: MongoDB provides advanced functionality and there’s a cost. It is not open source like some of the other technology. It is assumed there is a budget to pay for some of these advanced features.

**Integration**: Integrating this solution with the rest of the tools such as reporting and analytics tool can be a challenge. Analytic tools can software such as Tableau or Power BI. It is with the assumption that these tools has a connector to our served data stored in MongoDB and Kafka.

**Authentication and Authorization**: Cybersecurity is an extremely important topic especially if we’re dealing with deployment to the cloud. This solution does not address authentication – who the user is, and authorization – what access do they have.

**Migration**: It is important to deploy the new infrastructure without impacting the current business. It is best to release changes iteratively while keeping an eye on issues. This solution also does not address unit testing, regressing testing, and load testing which are all important to a smooth transition.

**Staff Experience**: It is unclear how large the Medical Data Company IT staff is and how much of it is dedicated to support, software development, and networking. This solution assumes the staff has the size and ability to learn and support all these new technologies. If the staff is not of sufficient size and capable, there’s a risk the project will waste time and resources.

# Data Lake Architecture for Medical Data Processing Company

Diagram

Description automatically generated

# Design Considerations and Rationale

## Ingestion Layer

The combination of both Apache Nifi and Kafka has been selected to ingest different types of data. Although you can argue only one is needed, I think the two together offers much more.

Nifi has over 300 processors that will ingest data coming from databases, FTP servers, APIs, etc. Instead of writing code from scratch, we can leverage these already built processors. Why reinvent the wheel when someone else has already solved the problem. Nifi also provides an intuitive web based gui providing seamless experience between design, control, feedback, and monitoring. Nifi uses flow-based programming and supports powerful and scalable directed graphs of data routing, transformation, and system mediation logic. Nifi is made for scale with features that are highly configurable, high throughput, and designed for extension.

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications. Kafka makes use of topics that can store large amounts of data which is designed streaming and delivering real-time data. All this provided in a distributed, highly scalable, elastic, fault-tolerant, and secure manner.

Nifi will be used for batch processing and simple transforms. Kafka will be used for large data set which needs to be consumed and processed in real time.

Apache Sqoop and Flume was considered for the ingestion layer. However, both products are not as mainstream or as adopted as Nifi or Kafka. Nifi offers an extremely intuitive user interface that makes it easy to adopt to. Both Sqoop and Flume does not have the same offering. Amazon Kinesis was also considered but we wanted to go with an open source tool and not be locked-in with a vendor.

## Storage Layer

MongoDB will be used to house and store the massive data set of Medical Data Company. Mongo offers horizontal scalability with native sharding. Horizontal scalability means the database can be scaled by adding more commodity servers to the cluster. Sharding means we can strategically divide the data across the cluster to allow maximum performance in retrieving data. Mongo will easily handle a 20% YoY data growth rate.

MongoDB offers exceptional back-up and recovery tools. They offer continuous backup which means you can restore to any point in time. Traditional snapshot backups are available and backed up to the cloud service of your choice. On-demand backups are also available. Recovery from backup is as easy as a button click. For Medical Data Company, snapshot backups should be run daily, weekly, and monthly. Continuous backup should be available for the current day.

Metadata will also be stored in MongoDB. The type of data we will store includes the source of the data, the time the data was available, and the time that the data was loaded.

MongoDB supports both structured and unstructured data so we have the ability to store data as they are. For example, CSV files can be stored in a typical structured format but other data can be stored in a unstructured JSON format. MongoDB is extremely flexible and can adapt to any format ingested into the system. However, MongoDB is a document oriented database and does not support columnar data sets.

MongoDB Atlas offers built-in security controls for all your data. Enable enterprise-grade features to integrate with existing security protocols and compliance standards. Data is protected with preconfigured security features for authentication, authorization, encryption, and more. Mongo offers role-based access management with sophisticated access rules to control which users and teams can access, manipulate, and delete data. All network traffic is encrypted using Transport layer Security (TLS), with flexibility to configure the minimum TLS protocol version. Encryption for data at rest is automated using encrypted storage volumes. We can also enable automatic client-side field level encryption to encrypt sensitive data before it leaves the application and lands in the cloud.

SQL Server, HBase, Cassandra were also considered. However, columnar data is not a strict requirement. SQL Server historically is not a good scalable database. MongoDB is simply battle tested compared to HBase and Cassandra. MongoDB is popular to the point where Amazon has a clone called DynamoDB and Azure has their clone called CosmoDB.

## Processing Layer

Apache Spark is selected to process data. More complicated data processing that Nifi and Kafka cannot handle will be consumed and processed by Apache Spark. Spark uses RAM at each stage of processing and runs 10-100x faster than MapReduce framework.

From a workflow perspective, all batch, real-time, CDC processes will flow through Kafka because of its capability to buffer large amounts of data. Any data that needs additional processing will go through Spark. Spark can run on clusters with thousands of nodes and work well up to petabytes of data. Once data has been processed, it is written back into a Kafka topic for buffering. A Kafka-MongoDB Connector is then used to persist data from Kafka topic as a data sink into MongoDB. As the demand for data and processing increases, Spark will scale accordingly by simply adding additional nodes.

Spark can handle ad hoc queries and processing because of its performance. The ability to store the data and process it in ram makes it 10-100x faster than MapReduce framework.

MapReduce was considered as a processing solution but it’s slower compared to Spark and therefore not used. MapReduce uses disk at each stage of the processing and disks are slow at 100-300 MB/second.

## Serving Layer

Serving layer is the final data set in which is then presented to the application layer. The application layer can be reporting, BI analytic tools, or web applications. The type of data depends on the application itself. Usually for reporting and analytic tools, we’re storing some sort of aggregate data and perhaps over a specific time horizon.

MongoDB has built-in connectors to integrate with various applications. For example, if company is using Tableau for BI analytics, there is a MongoDB/Tableau connector that can be used. As shown in the diagram, using files as a serving layer has been removed. Files are error prone, static, and not scalable. Previous applications that read from those files must integrate with MongoDB directly for the data.

Kafka is also used in the serving layer. Kafka is such a versatile tool that one can argue it is an all-in-one solution. For real-time streaming low latency streaming applications, the solution is to pipe directly to Kafka for the data.

If the company wants to add new services such as machine learning to their architecture, they can simply pipe into MongoDB or Kafka for the data.

# 8. Conclusion

Medical Data Processing Company is dealing with an extremely difficult data problem. A variety of big data technologies must be deployed to tackle this problem head on. Apache Nifi with over 300 processors out of the box can deal with ingesting a variety of inputs. Apache Kafka is a scalable, fault-tolerant, publish-subscribe messaging system that will act as the glue and will enable all the other systems to work together. Apache Spark will allow for highly distributed processing power across any dataset. And finally MongoDB will provide scalable storage across clusters while offering sharding for performance.

# 9. References

Real-Time Stock Processing With Apache NiFi, Apache Flink and Apache Kafka

<https://www.youtube.com/watch?v=sTnyu3fRUwE&t=546s>

Data Lake Architecture: How to create a well Designed Data Lake

<https://lingarogroup.com/blog/data-lake-architecture/>

Data Lake Architecture: A Comprehensive Guide

<https://virtasant.com/blog/data-lake-architecture>

Data Lake: Concepts, Characteristics, Architecture, and Case Studies

<https://www.alibabacloud.com/blog/data-lake-concepts-characteristics-architecture-and-case-studies_596910>

What Well-Designed Data Lake Architecture Looks Like

<https://www.xplenty.com/blog/data-lake-architecture-guide/>