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 | FirstName LastName | 0.1 | 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 |
| Pouya Ataei | Data Architect | Medical Data Processing | Pouya.ataei.7@email.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

The purpose of this document is to delineate the new data architecture designed for DataProcessingCo. This document contains the purpose, the requirements, the design, the assumptions, and the rationale behind the design. I’m creating this document to effectively communicate the properties of this new data design and justify its usage to solve some of the current problems.   
  
This is document is targeted for enterprise architect, data architects, data engineers, software engineers and technical directors. In-scope items of this document are purpose, requirements, design, assumptions and the rationale. Anything other than these, and out-of-scope items are detailed system specification, development process, and business process and workflows.

# Requirements

Some of the important requirements of the new data lake architecture is to improve up-time, reduce latency, increase system reliability, introduce horizontal scaling, and having a metadata-driven design. Current data architecture is inflexible and overly centralized using vertically scaled relational databases at its core. This architecture has been challenging to scale and maintain.

# 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

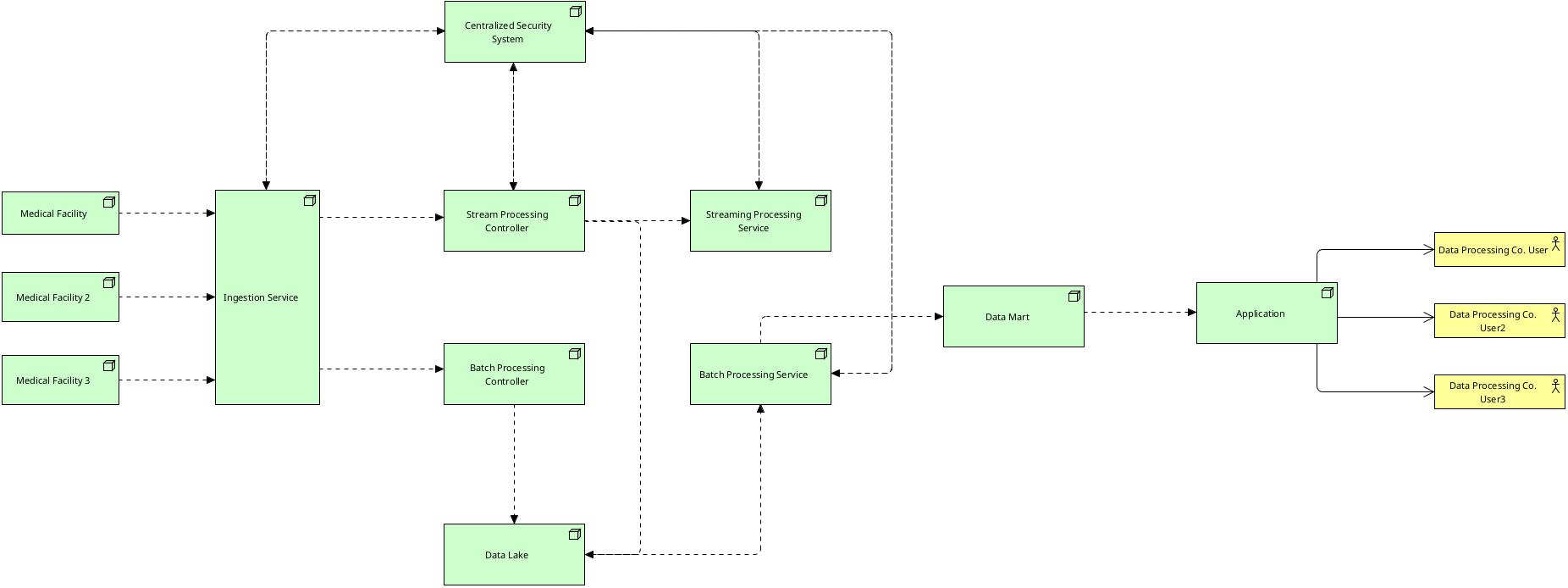
# Data Lake Architecture design principles

* Scalability: The architecture should be able to accommodate growth in the volume, variety, and velocity of data.
* Flexibility: The architecture should allow for changes in business requirements and technology advancements.
* Maintainability: The architecture should be easy to maintain, with clear documentation and modular components.
* Integration: The architecture should support seamless integration with other systems and technologies.
* Data Quality: The architecture should ensure the accuracy, completeness, consistency, and reliability of data.
* Security: The architecture should provide secure access to data and protect sensitive information.

# Assumptions

* There is a need for both batch and stream processing and therefore there should be separate services associated to these two different processing models.
* Data should be stored in data lake for metadata management and bulk batch processing.
* Applications can retrieve their data from stream processing and batch processing services.
* There should be an ingress that retrieves data from the customers.
* Stream processing and batch processing layer should be separated.
* The ingress could be just a simple Nginx server that handles different requests and forwards them to different services based on the nature of the request.

# Data Lake Architecture for Medical Data Processing Company



# Design Considerations and Rationale <at least 3 pages>

## Ingestion Layer

There will be an Nginx web server that acts as gateway into the system. This server will retrieve requests and based on the header of the request decides to send it either to the stream processing controller or the batch processing controller. The stream processing controller is running an event infrastructure that will buffer and pass the data to stream processing service. The streaming controller could be a Kafka instance, and stream processing service can be a Spark cluster.   
  
For the batch scenario, data is first stored in data lake through the batch controller, and then retrieved by the batch processing service. From there on, relevant data is retrieved by the batch processing service which processes the data and stores in a data mart. Hive can be used for batch processing and data lake can be Amazon S3.

This ingestion layer scales well, as the stream processing service is a kafka cluster, which is distributed in nature. As the demand increases, the number of brokers increase. In case of batch processing, S3 object store can be scaled up and even glacier classes can be used to save costs. Hive can run queries in a distributed manner too, so there should not be any bottleneck.   
  
I’ve considered Apache Flume and Sqoop. Howbeit, Kafka still is a better technology as it can handle streaming, pub-sub and basic event messaging too. Sqoop could be used if the ingestion heavily relied on relational databases, and Flume could be used if streaming wasn’t part of the future vision of the CTO. One could also look into service discovery, data dictionary, and schema registry.   
  
Moreover, Apache storm was considered for streaming. While this technology could effectively handle stream processing loads, Kafka is found to be a better option because it has a lower latency, has got built-in fault tolerant mechanisms, and is horizontally designed and scaled.

## Storage Layer

All data will be stored as is in the data lake. As the volume increase, so is the number of objects in S3. This is scalable as S3 is a distributed scalable object storage.

Data lake will be snapshotted daily, which can be used to restore data in case of event failure or data loss. Backups will be stored in glacier classes for reducing costs.

Metadata will be managed through Atlass centrally. Atlas will also help with relationships, data lineage and business glossary terms.

An ideal format is Parquet, that’s due to the fact that Parquet can compress approximately 70% better comparing to alternatives such as CVS and JSON, it is a colum-oriented data structure, and it has mechanisms such as predicate push down which allows for even more compression and smaller file size.

Apache ranger can be used as a centralized security administration. Apache allows fine grained authorization, role-based access controls, attribute-based access controls and control auditing. This service can handle authentication and authorization flows, as well as auditing.

Along the lines, I’ve looked into Apache Knox, and Apache Atlas. While Apache knox could be chosen for the ingestion gateway, it was decided that a more mature centralize security is a better solution.

One of the challenges of this architecture can be the complexity of spawning up a new Kafka broker in case of increased load and balancing again between brokers and consumer groups.

## Processing Layer

Depending on the nature of the load and the requirements (batch or streaming) data will be processed differently. Batch requests will go in to the batch processing controller and from there to the data lake and then to the batch processing service which uses Spark to process data. This can additionally allow for on-demand query if needed.   
  
If stream processing is required the data is flowed from Kafka into the stream processing service. This service is another Spark instance configured for stream processing. The reason the two are segregated is due to various needs in scale, latency, processing model and data volume. Maintenance can also be improved by configuring each Spark instance exactly to the requirements of the data it’s processing.

Ad-hoc query can be achieved by connecting directly by using data mart 2 designed specifically for data science and business intelligence workloads.

I’ve considered Hadoop HDFS and EMR, but S3 comes with more features out of the box. Using S3 means the implementation is freed from any HDFS specific technology such as HBase or Hive.

## Serving Layer

Serving layer in this architecture is designed as two purpose specific components, that is data mart 1 and data mart 2. Data marts store specific relevant data for specific data consumers, and this in turn increase performance, user autonomy, data security and governance.   
  
The data mart 1 is used by the application, which incorporates the schema necessary for application to easily consume and visualize and report on the data.

Data stored in data marts is usually relational structured data, but this can be augmented with semi-structured and unstructured data too.

Data marts will provide their APIs or drivers to third-party softwares and applications to communicate.

# 8. Conclusion <approx 2-5 lines>

<Conclude the contents of the document. Provide recommendations on next steps if any.>

# 9. References <If any>

<Provide links of any external documentation, wiki, blogs that you used to complete your research to put this solution together>