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	Michal Janczyk	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
Michal Janczyk	Data Architect	Medical Data Processing	michal.janczyk@olx.pl

1. Purpose

This document provides an overview of an implementation strategy that will be working at Medical Data Processing Company where data teams combine forces to publish a new solution. The main objective of the project will be to implement a new Data Lake that will replace the current solution.

The purpose of this document is to present all the technical solutions that will be used throughout the pipeline and to justify their selection. The document will contain all the decisions that were made during the validation of the technologies used, which in the future may help to avoid mistakes that could have been made during the first implementation of Data Lake. The main audience for this document will be highly qualified technical people with whom all the details of the changes to the current solution will be discussed as outlined below. The technical staff will be responsible for the maintenance of the infrastructure necessary for the operation of Data Lake and its proper functioning.

The basis of the work performed is the creation of a fully operational Data Lake and the transfer of all the knowledge gathered to the technical team, which will be responsible for the proper functioning of the solution on the client's side. The scope of work does not include subsequent technical support for the work carried out. All consultations will be subject to evaluation and pricing.

2. Requirements

Valid understanding of the data lake is being away for Medical Data Processing Company to collect, store, and analyze all of their data in one place. Nowadays, cloud computing takes center stage and legacy technology models fade into the background, this new paradigm is revealing its potential which can be used in this project. This will allow us to create modern cloud solutions, and make it an innovative, cost-effective, versatile data lake.

To make it truly useful, a data lake must be able to store data in native formats, facilitate user-friendly exploration of the medical data, automate routine data management activities, and support a broad range of analytics use cases. Most of today's data lakes, however, can not effectively organize all of an organization's data.

The current technical solution implemented by the Medical Data Processing Company is not sufficient to meet its needs. The company has experienced very high growth over the last few years. As a result, the volume of data is also starting to increase dramatically - which our customers are not ready for. The current data warehouse processes around 250 GB of data per month. The number does not seem to be great, but if we base it on an instance of one master code that concentrates all the data on itself and which is no longer able to grow horizontally, we come to a dead-end situation. An additional complication is the processing of data in real-time, which is also not possible in the current situation. The company manages this by processing data only at night. Increasing the hardware capabilities of the server on which the current solution is running also did not bring the desired results. Besides, the company uses an RDMS mode for the entire Noda which assumes indexes on rows which is no longer effective for large data warehouses. A temporary solution to the problem would be to put indexes on columns instead of rows. However, this would only delay the situation we are currently struggling with. The current framework is also unable to cope with the increased influx of data from external providers. The last time such a situation occurred the entire database server had crashed for several hours. This is unacceptable if we are going to process patient data from medical facilities that could, at a critical moment, decide someone's life or death. The engineering team rightly suggested reducing data on the server, but this is also a short-term solution. The existing solution also does not allow the architecture to be quickly set up from scratch and the data flow to be resumed in the event of an error in the system. Extra ETL processes are run every night, SQL Server Integration Services (SSIS) which are a component of Microsoft SQL Server 2005. These processes cannot work on different file types, so a single script per file format is needed. We will try to implement a native and generic (for all file types) solution based on AWS cloud which will replace the current scripts. The general problem is the performance of the database or rather the lack of it. Low performance due to high server load causes a lot of problems in data utilization by dedicated people. Therefore, our goal will be to create a reliable and stable infrastructure that helps solve problems instead of creating them. All the technical documentation, including the existing infrastructure and its problems, was described in detail in a document provided by the client.

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

3. Data Lake Architecture design principles

Modern cloud data lakes allow them to capture, store, and facilitate analysis to discover trends and patterns.

Our Data lake will store newer semi-structured data types in their native format, without requiring a schema to be defined upfront which is a huge benefit. With a modern, cloud-built data lake, we will get the power of a data warehouse and the flexibility of the data lake, and no limitations of both systems behind. Additionally, we also get the unlimited resources of the cloud automatically, scheduling and orchestration. The cloud data lake provisions the necessary infrastructure and platform services immediately, often using the code as an infrastructure paradigm. These implementations are not only less expensive and easier to scale, but they're also virtually trouble-free. Security, tuning, and software updates are automatic and handled by the cloud vendor, which is a very convenient method of upgrading our stack.

Designed data pipeline will continuously, automatically, and asynchronously detect new data as it arrives in our cloud storage environment and then continuously loads it into the data lake.

With the cloud infrastructure, we will be able to data-sharing technologies that have emerged, enabling organizations to easily share slices of the data, and receive shared data, in a secure and governed way. This way of working simplifies data sharing across the organizations.

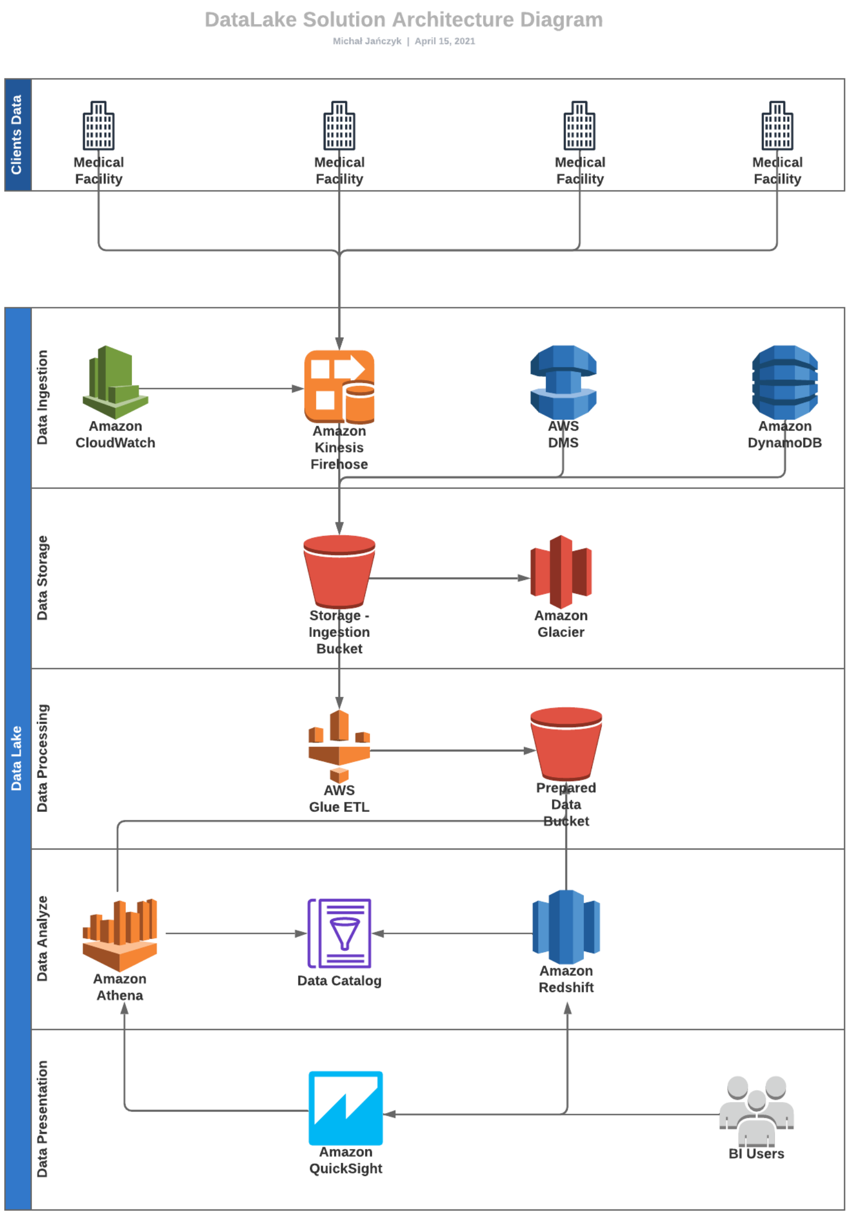
What is more useful is the multi-tenant cloud-built data lake enables our clients to share live data and receive shared data from diverse medical facilities without having to move that data. And there's no contention or competition for resources.

The designed cloud data lake will deliver all the resources that our client needs, with the instant elasticity to scale with demand. There is no longer a need to have over-provision resources to meet peak demands at the time, and cloud storage typically costs only a fraction of standard node monolith traditional systems. Client staff meets demands for data volumes, velocity, and variety on a scale. A very important aspect used by analysts is the SQL abstraction layer and it's a simple way to connect to the data in its raw format when they want to interact with it and don't have to move it from place to place. The process of copying data can be very demanding and costly. The proposed solution involves pointers to external tables and, ideally, materialized views External tables store file-level metadata about the data files such as the file path, a version identifier, and partitioning information. This enables querying data stored in files in a data lake as if it were inside a single database. To meet the customer's expectations, the cloud-built data lake can automatically scale concurrency by transparently creating a new cluster and then automatically balancing the load when needed. When the load subsides and the queries catch up, the second cluster automatically spins down. Our chosen cloud provider allows our client to dynamically expand independent resources to handle sudden concurrency issues. Ultimately, a data lake should democratize access to the data. People of all skill levels including line-of-business managers, financial analysts, executives, data scientists, and data engineering specialists should have easy and ready access to the system to perform the analytics they need.

4. Assumptions

1. The original promise of the data lake remains: a way for organizations to collect, store, and analyze all of their data in one place.
2. What if there will be a need to reprocess data due to broken pipelines?
3. What exactly will be the cost of the proposed solution?
4. Why not decide to implement the simplest solution - AWS Lake Formation?
5. How to secure your data lake with the role- and view-based access controls.
6. Problematic ways of having a large number of small files in a data lake (rather than larger files optimized for analytics) can slow down performance considerably due to limitations with I/O throughput.
7. The amount of effort required to include a data source into the data catalog should be invariant of the data’s size. Whether it's a tiny text file or a multi-petabyte Hive Table, it should only take you minutes to include it in the catalog.
8. Instead of having a cross-domain team when it comes to data, create a polyglot team that has members who can work with ingestion, Spark, and Kafka.
9. Although it centralizes just the names of various datasets and makes them easy to find, the catalog must also make it easy to access the underlying data if you need to.
10. Customers data doesn’t require a lot of low-level programming to implement patterns that are essentially decades-old – ingestion, transformation, and loading instead of using SISS.

5. Data Lake Architecture for Medical Data Processing Company



6. Design Considerations and Rationale

a. Ingestion Layer

We are planning to ingest different types of data by using - Kinesis Data Firehose. It is a service that Amazon offers as a part of AWS that is built for handling large-scale streaming data from various sources and dumping that data into a data lake. Amazon Kinesis Data Firehose is the most efficient way to load streaming data that comes from medical facilities into data stores and analytics tools. It will capture, transform, and load streaming data into the place where our client will need it the most - Amazon S3, Amazon Redshift, Amazon Elasticsearch Service, and Splunk, allowing the teams for near real-time analytics with existing business intelligence tools and dashboards which they are already using today.

A very useful feature for our client would be, the transformation of the data happens completely serverless, and without needing a complex pipeline setup. It is a fully managed service that automatically scales to match the throughput of our client medical data and requires no ongoing administration. Firehose was designed to be scalable right after the implementation on the client-side. There's not anything we need to do as users to make Firehose scalable, and because it's completely serverless, we don't even have to maintain EC2 instances, which will decrease our cost significantly.

It can also batch, compress, and encrypt the data before loading it, minimizing the amount of storage used at the destination and increasing security.

The second element of our data ingests stage will be - Relational Databases (RDS). Relational database systems used to be the core of most enterprise data systems. Our data lake will receive medical data from the relational databases on a periodic, scheduled basis, either through a data stream such as Kinesis Firehose, 3rd-party tools like Sqoop, or Apache Flume. We would like to pass the data through change data capture (CDC) using Amazon Database Migration Service. AWS DMS is a cloud service that would migrate relational databases, data warehouses, NoSQL databases, and other types of data stores that are currently in use. We can use AWS DMS to migrate all data into the AWS Cloud, between on-premises instances (through an AWS Cloud setup), or between combinations of cloud and on-premises setups. These data changes will be automatically pushed into the ingestion buckets in the data lake for later processing at the storage layer.

On the same basis, we are going to use Amazon DynamoDB which helps our client capture high-velocity data such as clickstream data to form customized medical facility equipment profiles to develop our internal BI and analysts fulfillment, improve their satisfaction, and get insights into important data that can decide on someone's life. It's essential to store these data points in a centralized data lake, which can be transformed, analyzed, and combined with diverse organizational datasets to derive meaningful insights and make predictions.

Third-Party Data Frequently might be used internally in our client organization. Data that comes from outside the organization will be valuable to integrate into the data lake. In this example, initial data from our client facilities curated by a third party will be included in the data for analysis. The data will be staged into S3 during startup by the CloudFormation script, but this data can be sourced through a variety of channels.

b. Storage Layer

What makes our cloud data lake special is the ability to store data in one location or multiple locations, having an integrated cloud analytics layer reduces risk and makes life simpler. We don't have to move or copy data among multiple physical data marts, and what it does is decrease multiple potential points of failure. The entire environment can be queried as a single source of truth via SQL and other familiar tools.

This separation of data and compute is a foundation of architectural concepts in modern data lakes implementation. By using S3 as the storage tier and Glacier as its backup, we can have transient data warehouses or Hadoop clusters that scale up to the compute capacity when you need them.

Amazon S3 is the most popular object storage built to store and retrieve any amount of data from anywhere. What also makes a perfect fit for our internal use is an object storage service that offers industry-leading scalability, data availability, security, and performance. This means customers of all sizes and industries can use it to store and protect any amount of data for a range of use cases, such as data lakes, websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. It is designed to deliver 99.999999999% durability and stores data for millions of applications used by market leaders in every industry. We could not have found a more suitable and realistically low-cost place to store data.

S3 is the milestone of a data lake. It provides the storage tier for data at rest. Data in S3 can be queried in place using Athena or Redshift Spectrum, mounted to Hadoop with EMR, and loaded into Redshift, there is an unlimited number of options on how to use it, integrate, query it.

Another element that will be used in our pipeline will be Amazon Redshift is a fast, scalable data warehouse that is a very useful and efficient tool for any kind of analysis. It is simple and cost-effective to service all our data across a data warehouse and data lake. Redshift delivers ten times faster performance than other data warehouses by using columnar index storage, machine learning, massively parallel query execution, and high-performance disks. Redshift is also natively integrated with S3 to allow for high-performance parallel data loads from and to S3 into Redshift.

To accommodate all possible business needs of our client data lake we should be versatile enough to ingest and immediately query information of many different types and sources, which might be an issue for a single node database but not for our data lake. That includes and input unstructured data such as photos of x-ray scan files as well as semi-structured data such as JSON, Avro, and XML. It also includes open-source data types such as Apache Parquet and ORC, as well as traditional CSV and relational formats. In an ideal world, a complete data lake strategy should accommodate all types of data. Make them more generic. Including JSON, tables, CSV files, and Optimized Row Columnar (ORC) and Parquet data stores, but this is even more. We should think about how to store and analyze unstructured data, we may need a separate repository, such as an external object-store. A separate repository may also be necessary if we would like to have specialized needs for data sovereignty and data retention policy, or if we must comply with certain medical regulations.

Another key feature of our data lake will be encrypting data, which means applying an encryption algorithm to translate the readable data into cipher data, which is a fundamental aspect of security. Data should be encrypted when it is stored on disk when it is moved between a staging location for loading into the data lake when it is located within a database object in the data lake itself, and when it's cached within a virtual data lake - so basically at any level of our pipeline. Query results must also be encrypted when needed. End-to-end encryption will be the default protocol playing on the data layer, with security methods that keep the customer in control, such as customer-managed keys. The best and reliable solution for data lakes employs AES 256-bit encryption with a hierarchical key model rooted in a dedicated hardware security module. This method encrypts the encryption keys and instigates key-rotation processes that limit the time during which any single key can be used. Data encryption and key management should be entirely transparent to the user but not interfere with performance.

We must also not forget about metadata, which can be a very useful element of our project. Robust metadata service is a fundamental data architect concept of the object storage environment. External tables store file-level metadata about the data files such as the file path, a version identifier, and partitioning information. This enables querying data stored in files in a data lake as if it were inside a single database with high efficiency.

c. Processing Layer

The answer to this question, as well as the solution to some of the previously mentioned problems is the AWS Glue service, built on the foundations of the open-source Apache Spark and Hive Metastore technologies. Our main processing service will be AWS Glue. AWS Glue is a serverless data integration service that makes our data more user-friendly to discover, prepare, and combine data for analytics, machine learning, and application development. AWS Glue delivers all of the capabilities needed for data integration so that we can start analyzing our medical data and putting it to use in minutes instead of months.

AWS Glue is a serverless service, here we do not manage a fleet of servers that perform operations on behalf of the service, there are no virtual machines to which we can connect using SSH or RDP. Instead, we configure individual elements of the platform, which will perform specific tasks for us. When the tasks are not being performed, we do not pay for the computing power (the power is calculated in the PDU unit. 1 PDU = 4vCPU and 16GB of ram) but only for the previously-stored objects (database and tables).

Data integration is the process of preparing and combining data for analytics, machine learning, and application development. It requires multiple tasks, such as capturing, discovering, and extracting data from various sources which come from storage. To enriching, cleaning, normalizing, and combining data than loading and organizing data in

databases, data warehouses, and data lakes. These tasks are often handled by different types of services that each use different products.
As an alternative for AWS Glue, we could consider using Airflow combined with Spark (or Spark itself). Apache Airflow is currently one of the most popular tools for decentralizing cyclically running tasks in the form of a workflow/pipeline. We use code to define workflows and schedule their execution. There is a lightweight API as well as a graphical interface (WebUI) available, which makes it possible to visualize even quite complicated diagrams and monitor their operation, which translates into easier troubleshooting. There is a log, task history, and jinja templates that significantly extends the possibilities of the code. We have a single place where we store our code and its output - a log of how our tasks are running - this greatly simplifies things, as simply describing these workflows as code gives more freedom to maintain, version, and share our tasks. Apache Spark is an open-source, general-purpose cluster computing platform with in-memory capabilities for Big Data processing and APIs for Scala, Python, Java, and R programming languages: Scala, Python, Java, and R. In contrast - to the two-stage, hard disk-based MapReduce processing engine used in Hadoop - the multi-stage processing used in Spark is based on in-memory technology. This allows most computations to be performed directly in memory, which is why the performance of this platform is often much higher than in Hadoop. In some applications, e.g. iterative algorithms for interactive data mining, performance differences can reach up to 100 times in favor of the Spark platform (see table for comparison).
Our choice, however, was AWS Glue. It provides both visual and code-based interfaces to make data integration easier. The Engineering team will easily find and access data using the AWS Glue Data Catalog. Moreover, the AWS Glue Data Catalog contains references to data that is used as sources and targets of your extract, transform, and load (ETL) jobs in AWS Glue. This is where we create Jobs written in Python or Scala with which to perform our transformations. If we can't cope with writing scripts, AWS Glue will suggest code that we can easily transform for our needs. As part of ETL operations we can, for example, convert formats, change table schema, parse or delete data, create new tables in a completely different location. To create your data warehouse, we must catalog this data. The AWS Glue Data Catalog is an index to the location, schema, and runtime metrics of our client's data. Data engineers and ETL developers can visually create, run, and monitor ETL workflows with a few clicks in AWS Glue Studio. Data analysts and data scientists can use AWS Glue DataBrew to visually enrich, clean, and normalize data without writing code. With AWS Glue Elastic Views, application developers can use familiar Structured Query Language (SQL) to combine and replicate data across different data stores. The AWS Glue Data Catalog is Hive compatible so it can be used with Athena, EMR, and Redshift Spectrum in addition to Glue ETL. What is a very useful feature is that we can add table definitions to the AWS Glue Data Catalog in the following ways. Run a crawler that connects to one or more data sources, determines the data structures and their types. Then writes tables into the Data Catalog. What is more, we can run our crawler on a schedule, to keep data updated in case of change. We can also use the AWS Glue console to create a table in the AWS Glue Data Catalog manually. The whole process of user interaction consists of just three main steps. First,

you build the data catalog with the help of data crawlers. Then you generate the ETL code required for the data pipeline. Lastly, you create the schedule for ETL jobs. AWS Glue simplifies data extraction, transformations, and loading for every engineering user.

d. Serving Layer

The effectiveness of Data Analysis Insights depends largely on the proper presentation of data, including various tools supporting such analysis. The amount of data is constantly increasing, so presenting it in the right form is crucial. Due to the large increase in the use of Business Intelligence and advanced analytical technologies, its use is playing an increasingly important role in the success of a business. Increasingly, small and large companies are looking for effective software to visualize the data they have collected, and the systems themselves are rapidly changing and evolving to provide the user with the quickest and clearest way to present data.

The growing volume of analytical data is forcing the producers of analytical systems to create such methods of data presentation that they are in the place where the user needs them and do not require the intervention of IT departments, visible to the user. An example of this is the built contextual AWS Quicksight system.

The complete data presentation strategy should accommodate all types of data, including JSON, tables, CSV files, or even XML files. The separate repository may also be necessary if we are going to have specialized needs for data sovereignty and data retention, or if you must comply with certain industry regulations that govern where your data is stored.

Amazon QuickSight is a lightweight, scalable, serverless, embeddable, machine learning-powered business intelligence AWS service built for the cloud and in the cloud. QuickSight will let us easily create and publish interactive BI dashboards that include Machine Learning-powered insights. QuickSight dashboards can be accessed from any device, and seamlessly embedded into our applications, portals, and websites. All delivered under one service provider. QuickSight is serverless and can automatically scale to tens of thousands of users without any infrastructure to manage or capacity to plan for. It is also the first BI service to offer pay-per-session pricing, where we only pay when our users access their dashboards or reports, making it cost-effective for large-scale deployments. Moreover, Amazon QuickSight can pull and read data from services like Amazon Aurora, Amazon Redshift, Amazon Relational Database Service, Amazon Simple Storage Service (S3), Amazon DynamoDB, Amazon Elastic MapReduce, and Amazon Kinesis. The service also integrates with an on-premises databases server, where we can file uploads and API-based data sources, such as Salesforce. QuickSight allows our end user to upload incremental data in a file or an S3 bucket. The service can also transform unstructured data using a Prepare Data option. An AWS user can select the data fields to be analyzed and then either drag them onto the visual canvas or allow QuickSight to determine the appropriate visualization based on the selection data.

8. Conclusion

Traditional data warehouses might be not efficient due to store data on database clusters both store the data and perform the operations on the data. This leads to many challenges as the dataset will grow in the future. Do you optimize your resources for computing or storage? If so, how did you do it?

First of all, let us perhaps explain what is in the system's favor. It is simple to manage, what is the most important thing is it does not generate any cost in an idle state. This kind of data lake can be managed by Agile methodology.

By storing the data on Amazon S3, we have a very efficient and cost-effective service that can store virtually unlimited amounts of data. Around S3 is an ecosystem of tools that enable direct querying of the data, or high-performance loading into a database cluster. This also permits the creation of transient clusters that spin up to perform the computations and spin down when they are complete.

In brief. Amazon S3 will meet all storage requirements. AWS Glue will catalog and process our data so that it is ready for analytics while providing the required computing power. Amazon RedShift will perform the final analytics on already prepared data (Data Warehouse and BI).

9. References

AWS Documentation