Report 3

Deploy Data Lake framework Kylo into ICT Server

Le Nhu Chu Hiep

1. Introduction

The data lake emerges popular nowaday in the big data field. But since this is a new terminology and technology, most of resource paper and guider for building a real data lake is rare. The normal research often stops on theory and keys constrain. Therefore, it is hard for architect to imagine the shape for a data lake system and its eco-system. Although the concept and constraints is important, the lack of detail information causes many difficulty on forming the system shape as well as its implementation. Therefore, as many famous people said: "Standing on the shoulders of giants", following a pioneerer will be a first step on developing your own data lake system. And up to the moment this report is written, "KYLO" is only framework that I know is completed with well design and suitable for academic research. Despite of some limitation of a pioneer framework, the project is totally open source on github with a quite good deployment doc for them who want to try it out.

Some background information of the kylo project: Kylo is an enterprice-ready modern datalake management software platform for big data engines building on top of Hadoop eco-system include spark and hive. It combines severals apache technology together becoming a self-service to process data and provides metadata management, governance and security best practice. The "Kylo" is a play on a Greek word meaning "flow" and serves data as a flow in pipeline. This solution function includes data ingestion, data preparation, operation dashboard, global search on platform of web application. Kylo was developed by the child company of Teradata is ThinkBig and the design is compatible with Cloudera, Hortonworks, v.v.

2. Objective

In the first intended, I want to focus on the deployment process, configuration and summary of the difficulty during the deployment task because Kylo is quite complex in configuration. But I found out a research thesis [1] about this problem where the issuse was discussed quite clearly. Therefore, I decided to advance my job a bit and it will be:

- Although the thesis [1] provides a detail step to install and config kylo project, it is a bit out of date from the time it released, and one more point is that I try to run Kylo in docker container so it has some slighly difference. Then, I still mention about difficulty and solution for both the out of date issue as well as the Kylo container problem. But the detail of setup will not be mentioned since the user can find it out in the thesis [1].
- After that, I will dicusses about several core idea of this framework from my point of view, its pros and cons, then my strategy on building my own data lake project depending on this project (step by step).

3. Deploy Kylo in docker container

Firstly, the kylo is a greate open-source project that can be easily found on github (URL: https://github.com/Teradata/kylo). In there, I get the link to kylo document, Jiro to report issue and Google group link of dev team. And the kylo document gives a fully instruction of deploy requirement and setup step. Everything is seem to be perfect.

Until I discover that whole link points to the kylo installation packages died. Next 2 weeks after, I found out the reason in link: https://groups.google.com/forum/#!topic/kylo-community/zRmfTfPyNpg

In here, Matt.hutton told that:
"Teradata made the decision to discontinue Kylo in early 2019. Teradata had been the sole sponsor of the open-source project and offered paid corporate training, consulting services, and enterprise

subscriptions for support. The team was unable to immediately secure a new sponsor and devs have moved on to other projects. The last official release of Kylo was March 2019. The downloads of the RPM and sandbox are no longer available but you may still build the RPM (or TAR) from maven."

3.1 Rebuild kylo from source

Since I could not get the kylo installation package, there are 2 options now: find some image deployed kylo in docker hub or rebuild it from source. The first option is better, but unfortunately, I prefer second option than one, and it enables the flexible in tunning system also, then it is not too bad option. Then I use the instruction in kylo doc:

- Install maven (kylo project developed with maven)
- Run build phrase of project in root github with command: `mvn clean install`

3.1.1 Maven2 Access 501 Issue

Since the project requires access to url: "repo1.maven.org/maven2" but this repo change from http to https for security improvement.

Simple solution:
I nano to file "pom.xml" of root repository, change line "http://repo1.maven.org/maven2" to "https://repo1.maven.org/maven2"

3.1.2 Fail Test Issue

The environment of my container and of the dev team is totally difference. Then we do not expect the successful test during build time. And the fail happen during build the spark core.

Solution:
My install apache spark and the test spark is not the same version, so I can reinstall Spark. OR, the better way, I skip all test during build time. Then the new update maven build command will be:

```
mvn clean install -DskipTests
```

3.1.3 Authentication Issue

In the first time, I build and install kylo version 0.10.0. After several setup environment, It work, but I cannot logout or access to HDFS or anyresource of the program. It is caused because the kylo get "Authentication access denny". I tried a lot of way. In the darkest time, I had even uninstalled mySQL and installed Mariadb for suitable with jdbc that project use: mariadb-java-client-1.5.7.jar. But nothing work. Finally, I read Jira to find if anyone met same error. That is the time I know that the project is deprecated.

Solution:
After 1 night depressed, I came back to the first option, using the kylo image builded by "keven4ever" - a sweet guy and his git support me a lot. It work for many machine BUT NOT working in the usth-ict-server (where I deploy the kylo on). But the point is the kylo verison that image used: v0.8.0.1 and it work. So I get out the solution **REAL SOLUTION**: Instead of using v0.10.0, I downgrade to 0.8.3 (Why this version ? My feeling said "it will work"). Then the steps will be:

- jumping back to tag 0.8.3 in github

  ```
  git checkout tags/0.8.3
  ```

- Rebuild project from source

  ```
  mvn clean install -DskipTests
  ```

- Setup new package

3.2 Setup kylo from tar package in docker

3.2.1 Setup environment

The first problem when deploying the project is “systemd”. I try to deploy kylo in container from archlinux base image. Then it can not run “systemd”, that mean I can not run any services automatically.

Solution:
try to run all service from terminal. It is not imposible but it take of me 1 weeks. Frist, I need to install hadoop, hive, spark, elasticsearch, activemq, apache nifi from tar file so that I can config them to run from foreground. Thankfully, everthing run well. In the very first day, I used “screen” - a usefull terminal tool to manage all services that need to run on foreground. But now, I find out the way to all in background (most have background mode, but except “hiveserer2” that I use bash script of keve4ever to control it on background - I LOVE THIS GUY). Exception for mysql, I used the mariadb container in docker and bind address to kylo container. So that I did not setup too muck about it.

3.2.2 Setup kylo

After untar kylo package. I follow Manual Deployment Guide of kylo to setup. Unfortunately, the most of setup script in kylo requests system server to run - which is impossible to run in docker. Then it is impossible to generate running script for kylo.

Solution:
It is very lucky that the kylo setup script was written in bash script, so my job was read each setup script, then reconfig them to run in my container. It cause 1 more week to setup.

3.2.3 Config kylo environment

Since the kylo doc has a carefull instruction about this config. It was still too complex and ambiguous. In that time, I try google search, stackoverflow, doc and test it on docker. Most of time, I try to run kylo service, if it crashed, I read log and try to reconfig system. After 1 week nightmare with nano and log, I finally can run kylo-service v0.10.0 and log in. But the problem about authentication appeared and becomes a big monster that I cannot defeat.

Solution:
The time I met the authentication problem, I am depressed a lot. And final decision is come back to use the kylo image of keven4ever. But for some reason, this image crashed in usth-ict-server. So I read his repo in github and try to solve crash. Although I do not solve the crashed, but his repo is a treasure with the suitable configuration for docker running. Thanks that repo + half week, I am succefully deploy kylo v0.8.3 in usth-ict-server. I refer his github link in reference. And finally, right before writting this report, I found out the thesis [1] where you can follow to config kylo. So the step will be:

- Following thesis [1]
- If you deploy in docker and fall to running it, go to keven4ever repo.

3.3 Kylo problem

After running succeffully kylo service and access it from web app. I can test its function. It has a great building GUI with monitoring function. Using guide is friendly and simple also. But from my point of view, it has several issues:

- The user can not upload their data through internet directly (Service problem)
- The file size is small and multi-file ingest is slow (reported in thesis [1]) (Architect problem)
- Support structured or semi-structured only. The system do not handle the binary data (Architect problem)
- It do not split out the user and group access. That mean all user share same data ingest. (Security problem - Service problem)

NOTE Architect problem is the issue come from the design of system - it is hard to fix. On the other hand, the lack of some addition service in system called a service problem, it can be overcomed by developing additional plugin.

4. Kylo simple architect and Issues

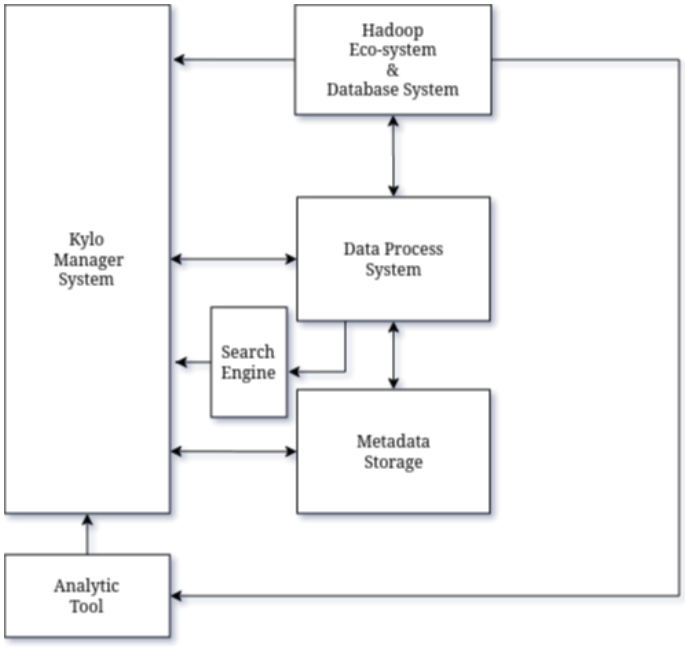

Figure 1: kylo simple arch

The idea befinds kylo is quite simple, the system focus on self-service and data flow. The system allows user to scketch out the flow of data. Then the data process system uses that template and self-service process the data. Although the "data process system" action depended on the schema defined by user, the basic job of the system will be:

- Whenever the data ingested, flowing it through a pipe
- In the pipe, data will be transform and auditing, the log recorded for provenance
- Finally, data will be stored in HDFS and the log file will through to metadata storage
- The index information will be updated in search engine
- Next time, if the user requests data, they can access to metadata storage and HDFS to extract it
- And the system also provides a simple transformed tool on top of spark called "analytic tool" to extract data in a more formal form than raw

Since this system defines pretty well how a data lake system should be, it becomes a standard for understading and designing data lake platform. Though that, it comes with some architect issues that need more effort on designing to fix: Firstly, the kylo requires each file ingested must be audited and logged before storing. But instead of a parrallel process, kylo use the nifi process called "getFile" and serialize this task. That cause a bottleneck in whole system. Another problem is the auditting process of kylo. During the audit time, the kylo try to profile data (points out what data is valid and what is invalid). And the kylo leverages the apache nifi process for that job. It is considerd as a best solution from the kylo dev team point of view because it is enough to handle their normal data (mostly structured data). But for a general data lake, it will be a barrier for another developer with many different type and format of data that the kylo does not support. That mean whenever the user have new type of data, they must work with nifi and its mechanism. It reduces the flexibility of the system. Moreover, passing audit responsibility for a single process like apache nifi will waste the parallel power of a cluster. Although coming with several issues, the kylo has a great suitable architect for any data lake architector to leverage and refer.

5. Data Lake Design

5.1 Data Lake Redefine Definition

After spending effort on analyzing the kylo architecture, I realizes some misunderstanding of myself in data lake. Lets look at below picture:

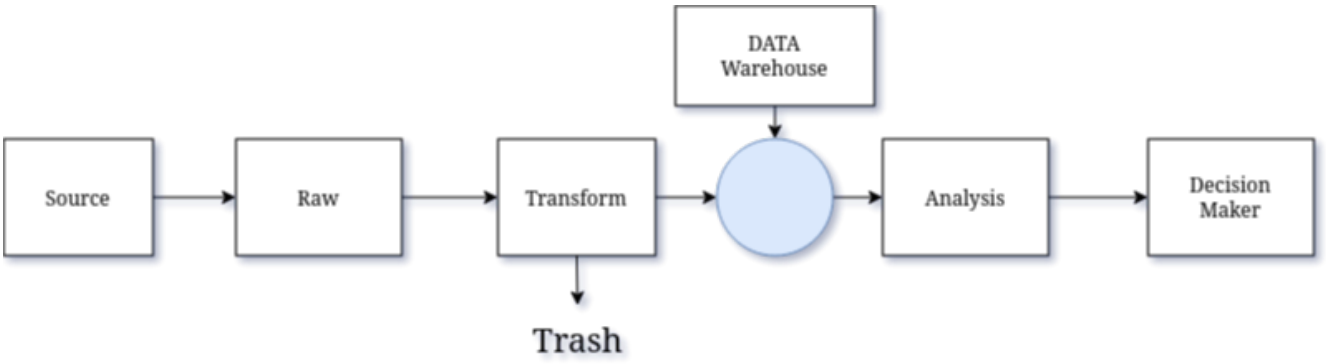

Figure 2: Data Warehouse Join Role

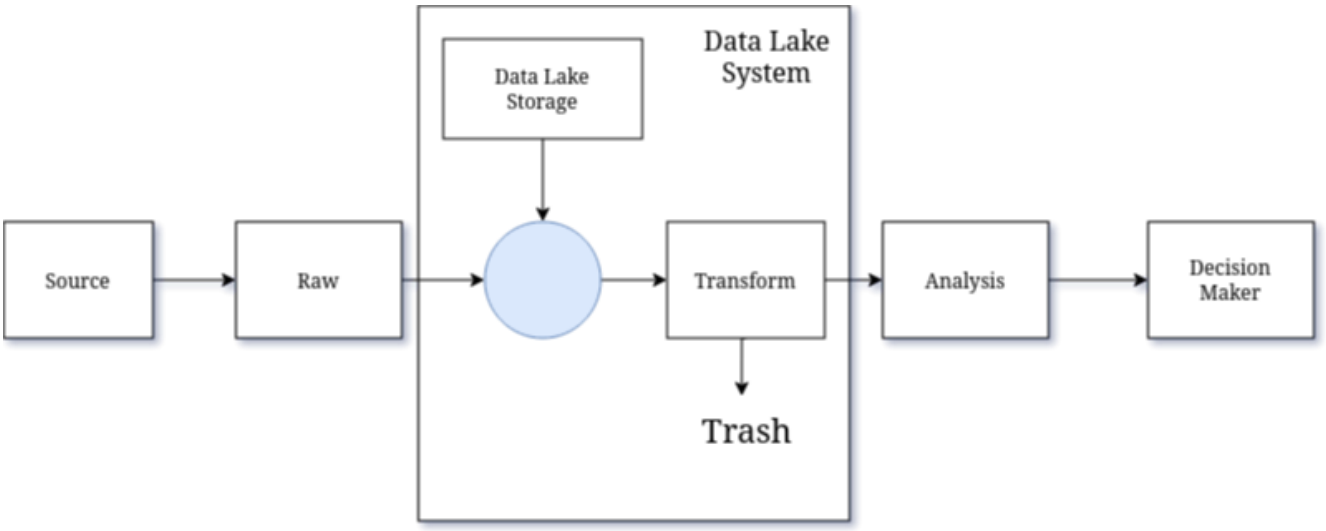

Figure 3: Data Lake Join Role

In here, the data lake does not replace the data warehouse, instead, it joins in different phrase and provides a bigger functionality than datawarehouse. In the pipeline of data, the data warehouse act as a queue between the transform phrase and analysis phrase. This queue keeps data growing large enough to feed into the analysis phrase. On the other hand, data lake is expected to not waste anydata. Therefore, instead of set queue after the transform phase, the queue push back after the raw phase. Then whenever a new decision maker requested, the data will push from queue to transform phase. But since the analysis phase using the analytic tool which is designed to handle a flow of a huge data at once, the data warehouse needs to cover only a storage queue. It is not true for the normal transform or ETL tools, it can handle a small flow of data at once. Therefore, when the data lake queue stand on middle of raw phase and transform phase, the transform tool required to be redesigned. In that case, the data lake system should cover also the transform phase. Finally, a data lake includes 2 part: a storage system and transform tool using big data technique.

5.2 Data Lake Component

Since the data lake is a centralize storage, it should have 2 basic parts: data storage and metadata storage. For the user data (big data) it is saved to data storage, and the extra information is passed onto metadata storage.

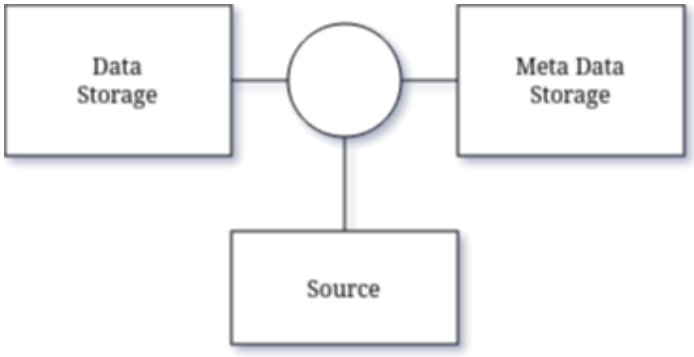

Figure 4: Storage and Metadata

The data storage provides storage space and the metadata helps you to audit and monitor information. Then we will want a mechanism of indexing data, a searching engine will be a greate option in this situation.

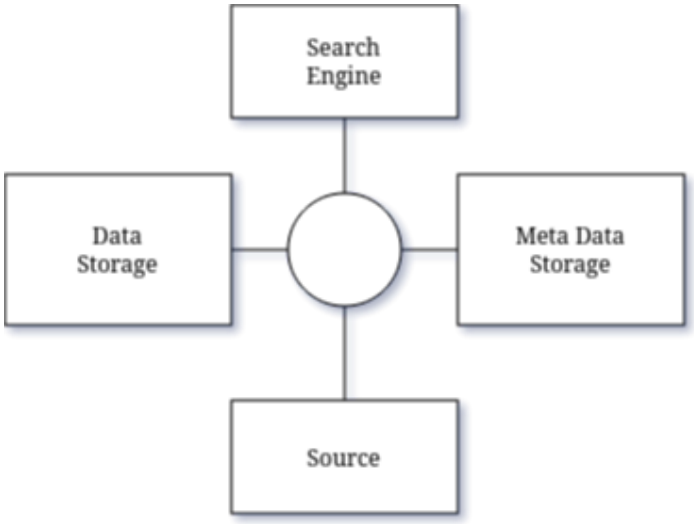

Figure 5: Include Search Engine

Great ! now almost important part of a simple storage management platform is done. Now, the system need a heart to run process. This heart can be divided into 2 part: data process and system controller. The data process should automatically get raw data from source, extract any needed information for profoling, auditing and indexing data, then flow all that information included raw data itself to the right position. Since acting as a self-service system, the data lake need a mechanism allowing administrator to keep tracking and controlling the storage. And that is the system controller. This component can access and controll whole system in an emergency case. Although the system controller developed for emergency situation, we can build several layer on top of its for more useful function. I split it into 3 layer: layer 1 (lowest) allow administrator to fully access, control and modify each component of system. Layer 2 (middle) enable the user to access storage but can not modify its, but the user can update and config the data process. In this way, the system accepts the user can indirectly modify another part of system through the data process. And finally, layer 3 (highest), the user can only access data and do not have any permission to modify the system.

NOTE this is only schema on paper, the actual implementation is depended on the dev team.

OOPS! Something missed in this design, that is the transform tool. Because the data lake covers also the transform

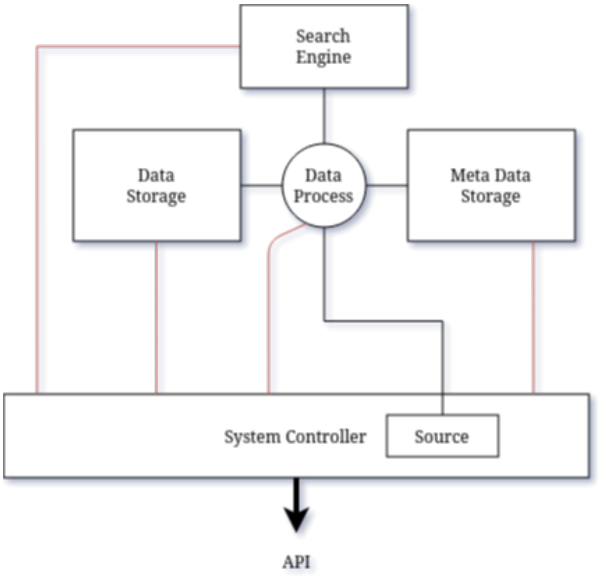

Figure 6: Data Lake Core Design

phrase in the pipeline of data processing, we expect some transform tool somewhere in design. Additionally, the system has not provided the user interface to the user to access to system and the way to ingest data into. The API is solution of these issues, the system controller will pass out its access through a bundle of API. For the ingestion, the system try to be simple as possible, so the source component is merged into they system controller and be pointed by API.

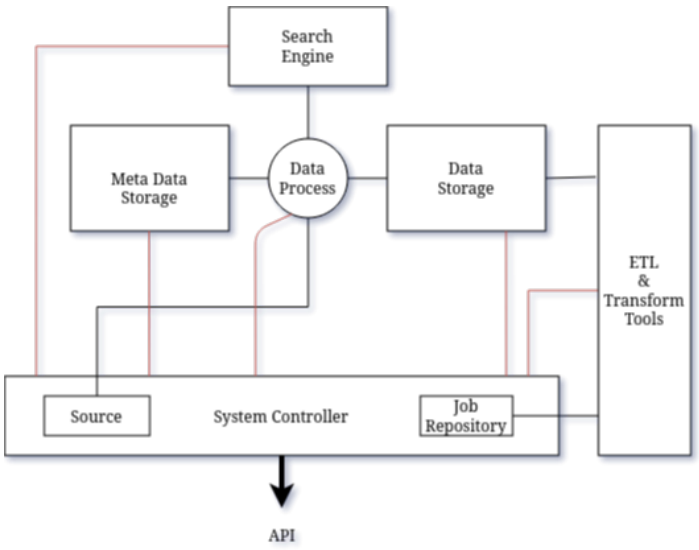

Figure 7: Data Lake Full Design

In here, the transform tool will pass result to job repository and controlled by System controller. So that should be a comprehensive data lake in my point of view.

5.3 Data Process detail

There are bundle of best practice on designing the data lake storage structure and provenence process (from the way of design partition to profiling process). But in the end, I recognize that a simple lake should have 3 main process: ingest, schema creator and audit. Ingest process reconstructs data from source path to a specific structure storing in

the cluster. Going with ingest process will be the schema creator process which are responsible to create a schema of metadata for a specific data domain. Both 2 processes called the prepare process. After that, the final process - audit process will run through out the already saved data and perform several profiling and tracking task to extract the valuable information following predefine schema by schema creator process for updating the metadata storage. Finally, for the searching purpose, each process should support a index task pass to search engine.

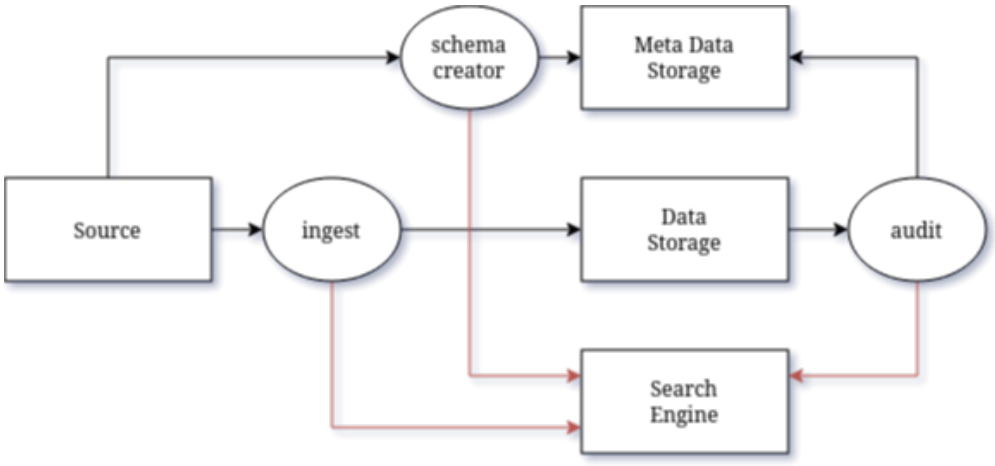

Figure 8: Data Process Detail

A good implementation of both 3 processes leads to a well data lake. But since the definition of a well data lake is very dependent on each client target and data properties. Therefore, if the user can use their own implementation on each process, it will help the system to become very flexible and high compatible with many environment. By contrast, the system consistency reduced because the process is manually implemented. Anyway this is only the theory idea, the true properties of system will be decided during implementation time.

6. HiViLake Discussion (Part 1)

The hivilake is the datalake expected to follow above architecture. But the architecture is too abstract to implement at once. So the process splited into 3 phrases:

- Phase 1: Solution for the data process part
- Phase 2: Solution for the system controller part
- Phase 3: API implementation and ETL component merging

Now, let begin with phrase 1, I separate it to more sub step:

- Step 1: Choose out the technology for storage system (meta storage and data storage)
- Step 2: Implement both 3 process: ingest, schema creator and audit concentrating on data flow
- Step 3: Find out a suitable search engine and merge it to the system

For the step 1, I have my own decision, the data storage will use hadoop 2 since this is the core technique of my research. The meta storage will be updated frequently and do not expected to keep tracking. So a system that handle good ACID task on a structured data will be a potential candidate. And the popular option that satisfies the requirement, at this moment, is mariadb system. Then, the core data process v0.1 should have a shape like:

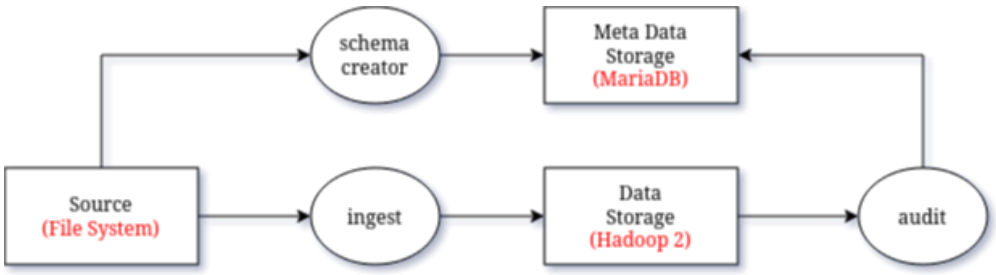

Figure 9: HiViLake Components

Next in step 2, the system requests 3 processes: ingest, schema creator and audit. Since these process provided by the user, there are several apache technology can be used for implementing. Then I will list out in below:

Tech	Type	Pros	Cons
Apache Hadoop	Distributed Storage	- Distributed File System - Scalable - Cost-effective - Multiple copies	- Lack of Preventive Measures - Small data poor performance - Risk security
Apache Spark	Distributed Computing System	- Fast Speed - Advanced Analytics - Dynamic in Nature	- No automatic optimization process - Small Files Issue
Apache Hudi	Data Management	- Upsert support with fast, pluggable indexing - Manages file sizes, layout - Timeline metadata to track lineage - Savepoints for data recovery	- Support structured data only (Parquet and Avro)
Delta Lake	Data Management	- Support ACID transaction on top of HDFS - Scalable Metadata handling - Time travel - Unified Batch and Streaming Source and Sink - Schema Enforcement - Schema Evolution - Audit History - 100% Compatible with Apache Spark	- Support well only with Apache Parquet

Tech	Type	Pros	Cons
Apache Zookeeper	Distributed Message System	- Simple Distributed Coordination Process - Synchronization - Ordered Messages - Speed - Scalability - Reliability - Atomicity	- No Migration - Data Loss potential - Kerveros unsupported - Complex maintenance
Apache Nifi	ETL tool	- Perfect implementation fo dataflow concept - Opportunity to handle binary data - Data provenance	- Lack of live monitoring and per-record statistics - Simplistic UI
Apache Solr	Search Engine	- Flexible and powerful query language - High-speed response query - Cluster mode supported	- Authentication non-supported - Zookeeper required - Master node requires reconfiguration if downing
Elasticsearch	Full-text Search Engine	- Scalability - Parallel Processing - Replacement MongoDB and RavenDB - Multi-data supported - Fault Tolerance - REST Api supported	- Only JSON supported - Hardware requirement

For the first time come across kylo framework, the apache nifi impressed me a lot on the self-service and flexibility of its mechanism. But It still a bit complex a simple lake in my imagination. Therefore, I decided to implemented whole process of HiViLake in pure hadoop and spark lib wiht java. Ofc, the hivilake is very flexible and supports any mechansim as long as they follow the specification to work with the system controller (It has not yet been built at this moment). Finally, My target has just stopped in the step 2 of phase 1. If everything work well, next step will be record in another report.

7. Conclusion

For the first time, my imagination about a data lake platform is quite ambiguous even after spending half-month on researching. But when I successfully deployed the kylo framework, discoverry alomost fully its architecture, the shape of a simple data lake was reset on my mind. Suddenly, everything is clarified, the lake techonologies and terms becomed easier to understand and make sense (like delta lake or apache hudi or stream processing). Then many idea of a simple data lake across my mind that formed the hivilake. Although there still have many thing to do, my outline of a system implementation was scketched out and I know exactly what I need to do. My point in here is the power of a sample. Sometime, a real example is equal to half or one month of reading the theory research. And kylo is a pretty greate sample for anyone who want to know the real shape of a data lake.

Finally, the next time is short about half-month only, and I have just finished about 1/3 of the job. That will be difficult time, but fun (maybe). I have a plenty of big data tool and techonology, and I need to assemble them into my lake. Any technology is also new with me, but I will try to build a pure lake with hadoop and spark and some mysql. Then if the time allowing, I will test a higher technology like delta lake and apache hudi or even apache nifi for completing the flexibility of the lake. By the way, my perfect lake should be compatible with any technology and easier replace a techonology with another if neccessary. Moreover, the lake should support any type and format of

data ingestion. It sound crazy at this moment, so for now I focus on Sound and DICOM data only and the core will be developed surround them.

One more thing, the philosophy of the lake will be: simple, flexible and replaceable.

Reference

[1] Kylo Data Lakes Configuration deployed in Public Cloud environments in Single Node Mode - Rong Peng

[2] Freecodecam_apache_nifi - URL: https://www.freecodecamp.org/news/nifi-surf-on-your-dataflow-4f3343c50aa2/#741e

[3] https://github.com/keven4ever/kylo_docker

[4] https://data-flair.training/blogs/13-limitations-of-hadoop

[5] https://hudi.apache.org/docs/comparison.html

[6] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6033252/

[7] https://news.ycombinator.com/item?id=19741271

[8] https://freshcodeit.com/freshcode-post/top-5-enterprise-etl-tools

[9] https://www.courseya.com/blog/what-is-elasticsearch-pros-cons-and-features-list/