Report 4

Hivilake Logical Design

Le Nhu Chu Hiep

1. Introduction

This is a report in the hivilake project document about architectural design of the lake. Since most of the modern data lake is concentrated on RDBMS and NoSQL. And the requirement for a simple lake to efficient store the large binary file has not yet satified. Therefore, the first purpose of hivilake design try to solve that issues. But for this report, only the general logical diagram of system is mention and the explanation about mechanism of managing the large binary file can be discussed on another report.

2. Objective

In the document, we show out the solution for storing and managing the binary file. Simple design and easy to implement is our highest priority. So the document mention about only 2 importance parts with their explanation:

- Usercase list
- Logical diagram

3. Methodology

3.1 Usercase list:

For the convenient of design as well as implement tasks, we list out several usercase satisfy a basic demand of anyone who prefer to store and manage their big data.

List of usercase:

- Auth:
  - Login / Logout
- Basic User:
  - User see their profile:
    - username
    - password
    - name
    - gender
    - role
    - department
    - email
  - User edit their profile
  - User change their password
- Admin
  - Admin create new repo
  - Admin modify description to repo
  - Admin mofify expected properties of file in repo
  - Admin create category (label / tag)
  - Admin modify category description
  - Admin create new user

  - Admin add/delete/update the “force condition” to user
- Feed (Ingest)
  - Admin create new feed
  - Admin add user to feed as Miniadmin/Editor (from right -> left: user permission increase)
  - Miniadmin link feed to repo
  - Miniadmin update feed properties (description, category, file_description)
  - Editor push file to feed
- Monitor
  - User can monitor following repo
- Search
  - User see the list of repo, list of label
  - User search file by repo name and extra properties (label, file_description, force condition, . . .)
  - User choose amount of file
  - User see detail of chosen file
  - User copy chosen file to their local

3.2 Logical diagram

From the usecase diagram, the lake machenism to store and manage the information will be:

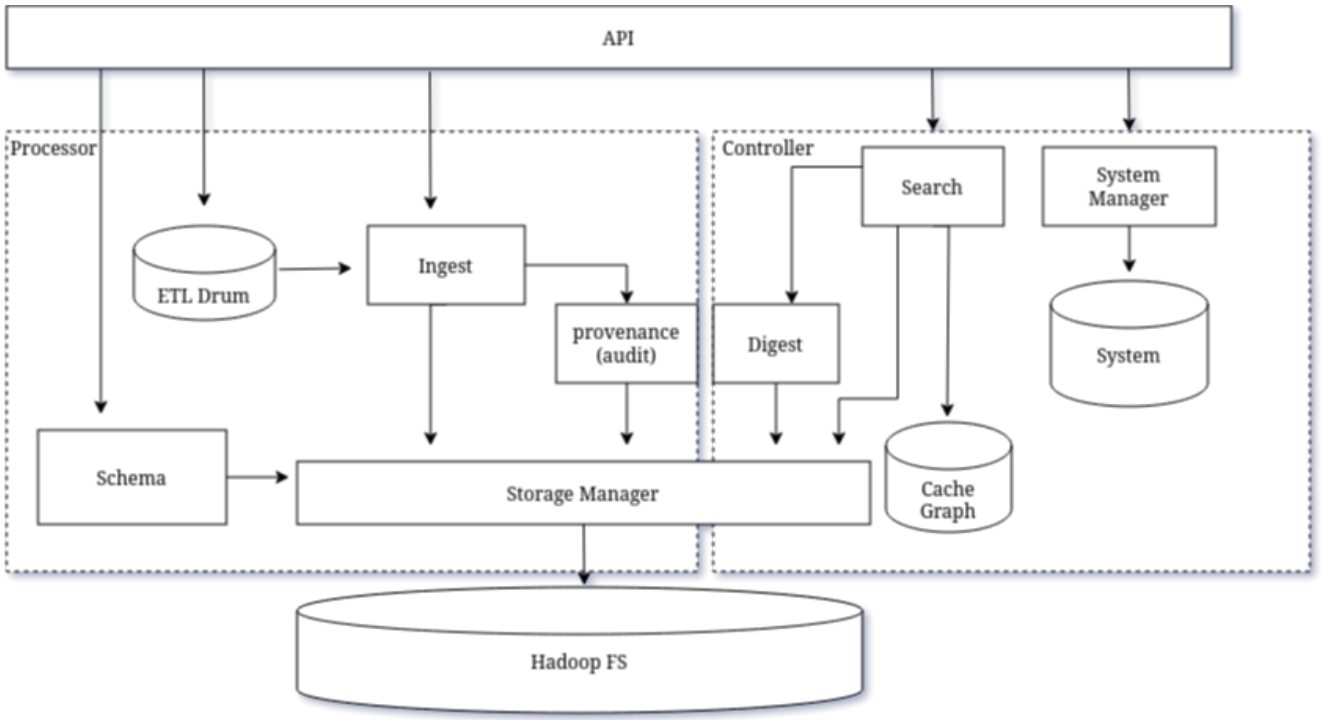

Figure 1: Logical Design

This diagram is not represent fully feature mention in the usecase. The main point in this diagram focus on the core of lake only and provides a framework to manage the lake. In the system point of view, it is divided into 2 sub-systems:

The processor system is responsible to create and update file in storage. For now, it has 2 main block:

- **Schema**: logical block to create repo
- **Ingest**: logical block to push file into repo, update metadata and provenance data in repo. For provenance task, it use a sub logic called **Provenance (audit)**.

The second system is controller. In here, this block system enable use access to lake, search file and get out data. Moreover, the system metadata is controllered in here also. Therefore, the sub-system has 2 main block:

- **Search**: logical block allow the user can search and extract data from lake. It follows "lazy" mechanism - the search engine index and build graph data in the first time the user request. Then the graph cached into "cache graph". In the next time, if the search repo is not updated, the cache will be used. For degist data, the search call to the sub logical block **Digest** which will access and response back the corresponding binary file.
- **System_manager**: This logical block link to the system metadata. The api system will interact to metadata throught this callable block. The metadata (now) includes
  - category metadata
  - repo metadata
  - activity metadata
  - audit metadata

3.4 The system properties

This framework do not provide a fully automatic system. Instead, it request more effort on building the external system on top of hivilake. So, what the hivilake provide for the user ?

- The lake provides a simple mechanism to create and manage binary file.
- The lake allows a "lazy" search to discover the binary data.
- The lake supports an internal database to keep control, keep track and monitor the flow of data in system. But this database system is optional for hivilake.
- The lake prefer restful API for their external communicate. Therefore, it will be easy to integrate or develop a external system on top of lake.

The lake core should not support:

- Security feature.
- Authorization feature.
- User friendly interaction feature.

The lake non-functional properties:

- The system is simple for anyone to understand its abstract idea
- Since the core idea of hivilake is the self-describe repo. The logical block on top of it is easy to replace and update. So the system can be considered to flexible and replacable.
- The scalable and external is another properties of the framworke.
- The document of the system is public and have many figure to anyone who want to understand the abstraction idea.
- The code style and structure of the project is terrible but the author is a nice guy and pleasure to explain for anyone asking. Then, the project has a enthusiasm support author.

3.5 Work strategy

Job of developer:

- Le Nhu Chu Hiep (lazycoder):
  - New report explain the self-describe mechanism of the repo for binary storing.
  - Update feature for core framework.
  - Implement core of framework.
- Nguyen Viet Dung:
  - Design external system on top of hivilake following the usecase list.
  - Implement external system.

Reference

[1] https://spark.apache.org/docs/latest/running-on-yarn.html

[2] https://www.linode.com/docs/databases/hadoop/install-configure-run-spark-on-top-of-hadoop-yarn-cluster/

[3] https://hadoop.apache.org/docs/r3.0.3/hadoop-project-dist/hadoop-common/SingleCluster.html#Standalone_Operation