University of Science
and Technology of Hanoi

Bachelor Thesis

Development of Data Lake Core with Append-Only Storages and Query Polymorphism

Author
Nguyễn Gia Phong

Supervisor
Trần Giang Sơn, PhD

July 12, 2021

Contents

Declaration

I declare that I have composed this thesis in its entirety, as the result of my own work, unless explicitly indicated otherwise via referencing. The presented work has not been submitted for any previous application for a degree or professional qualification.

Lời Cam Đoan

Tôi xin cam đoan rằng tôi đã soạn toàn bộ luận án này, hoàn toàn từ kết quả công việc của chính bản thân tôi, trừ khi được chỉ rõ qua trích dẫn. Công việc được trình bày chưa bao giờ được nộp cho việc cấp học vị hay chứng chỉ trình độ chuyên môn nào trước đây.

1 Introduction

1.1 Motivation

Many researchers at University of Science and Technology of Hanoi (USTH) operate with data on a regular basis and often a dataset is studied by multiple researchers from different departments and points in time. Currently the data are organized manually, even on the laboratories' storages, which is prone to duplication and makes data discovery difficult.

A data lake shared among the university's researchers, professors and students will not only save resources but also improve productivity and promote interdisciplinary collaborations. With USTH's goal of growing to be an excellent research university in Việt Nam and in the region [1], building such data lake can be an essential task.

1.2 Background

A *data lake* is a massive repository of multiple types of data in their raw format at scale for a low cost [2]. The data's schema (structure) is defined on read to minimize data modeling and integration costs [2].

For the ease and efficiency of scaling, a microservice architecture could be a good choice. By arranging the data lake as a collection of loosely-coupled services, it becomes possible to scale individual services individually [3]. In this architecture, the *core* microservice is defined as the innermost component, which communicates directly with the storages. The core shall provide an application programming interface (API) for other components to upload, query and extract data.

Append-only storages only allow new data to be appended, whilst ensure the immutability of existing data. As immutable data are thread-safe, they reduce the complexity of the concurrency model, making it easier to comprehend and reason about [4]. This is particularly useful in large distributed systems with multiple moving parts.

Since the data are immutable, each *content* can be given an identifier (ID), i.e. a content ID (CID). For end-users, we also introduce a higher level concept: *dataset*, composing of not only the content but also relevant metadata for indexing. Like contents, datasets can also immutable, with changes written as a new revision linking to the previous one.

Append-only storages' operations boil down to two kinds: appending and reading. For the latter, sometimes the data are not wanted in their entirety, but filtered and accumulated. While data of different types usually requires different tools and libraries to query upon, the core API should be providing one single query language for all data types, plus their metadata. In this report, such usage is referred to as *query polymorphism*.

1.3 Objectives

The work presented here was done as part of a three-month internship in collaboration with several other students at USTH ICTLab[1] to build a data lake for a better management of the university's data. The internship focused on the lake's core microservice, which abstracts underlying persistent layers and perform relevant metadata transformation and discovery. It should provide an internal interface to other components for data ingestion, (primitive) query and extraction, as well as carrying out tasks for enhancing the discoverability and usability of the aforementioned datasets.

After the internship period, the resulting codebase shall be maintained by ICTLab and future students, so the work must be designed, implemented and documented in a way that ensures such possibility.

1.4 Expected Outcomes

The intended deliverables of the three-month internship are listed as follows:

- Requirement analysis of the data lake core
- Data lake core's architecture and design
- Core API design and specification
- Implementation and integration with other components

[1] `https://ictlab.usth.edu.vn`

2 Methodology

2.1 Requirement Analysis

In this section, from given context and objectives, we analyzed the expected system for a set of features and derived a list of use cases. Supplementary specifications were also added to elaborate on the nonfuntional requirements.

2.1.1 Use-Case Model

As previously introduced, the most basic functions of the data lake core are content uploading and downloading, along with datasets addition and querying. A more advanced (and rather powerful) use case is content extraction, which allows one to fetch only the interested part of the content, e.g. extracting rows matching a certain predicate from (semi-)structured data. Together with logging, the core's use cases are summarized in figure 2.1.

Upload content

This use case allows other microservices of the data lake to upload a content. Its flow of events is depicted as follows, where error handling is omitted for brevity, since all errors, if occur, replace the normal response.

1. A content is sent to the core microservice.
2. Core adds the content to the underlying storage and register it to the database (DB).
3. Core responds with the CID of the added content.

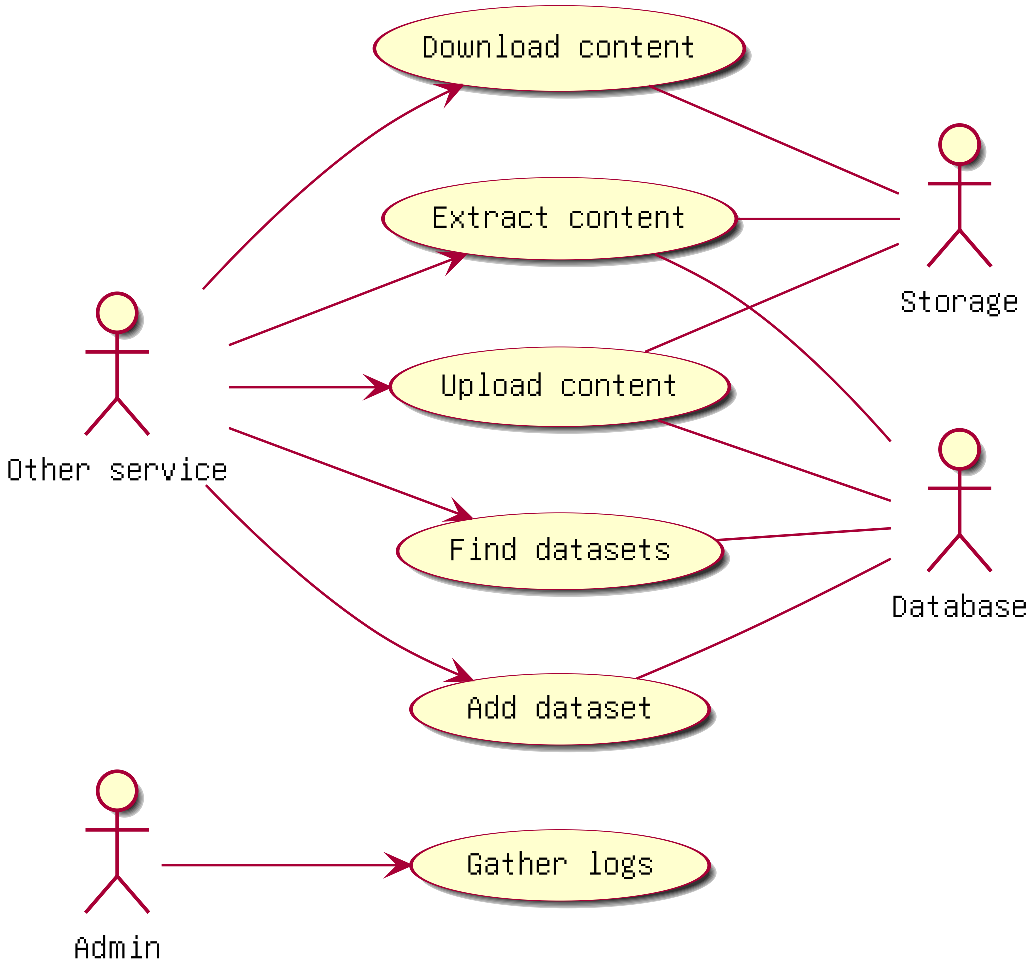


Figure 2.1: Use-case diagram

Add dataset

This use case allows other microservices to add a dataset:

1. A dataset is sent to the core microservice.
2. Core adds the dataset to the underlying DB.
3. Core responds with the ID of the added dataset.

Find datasets

This use case allows other services to find the datasets whose metadata satisfy a given predicate:

1. A predicate is sent to the core microservice.
2. Core runs a query in the underlying DB to find matching datasets.
3. Core responds with a linear collection of metadata, each of which satisfying the given predicate.

Download content

This use case allows other services to download a content:

1. A CID is sent to the core microservice.
2. Core passes the CID to the underlying storage.
3. Core responds with the respective content.

Extract content

This use case allows other services to extract a content's parts satisfying a given predicate:

1. A CID and a predicate is sent to the core microservice.
2. Core iterates the content for matching elements.
3. Core responds with the extracted elements.

Gather logs

This use case allows system admins to study events occurring in the core microservice for debugging purposes:

1. A system admin requests logs from core.
2. The admin receives the list of past events.

2.1.2 Supplementary Specification

2.2 Architecture

2.3 Design

2.3.1 Technology Choices

2.3.2 Interface

2.3.3 Database Schema

2.3.4 Query Abstract Syntax Tree

2.4 Implementation

2.4.1 Error Handling

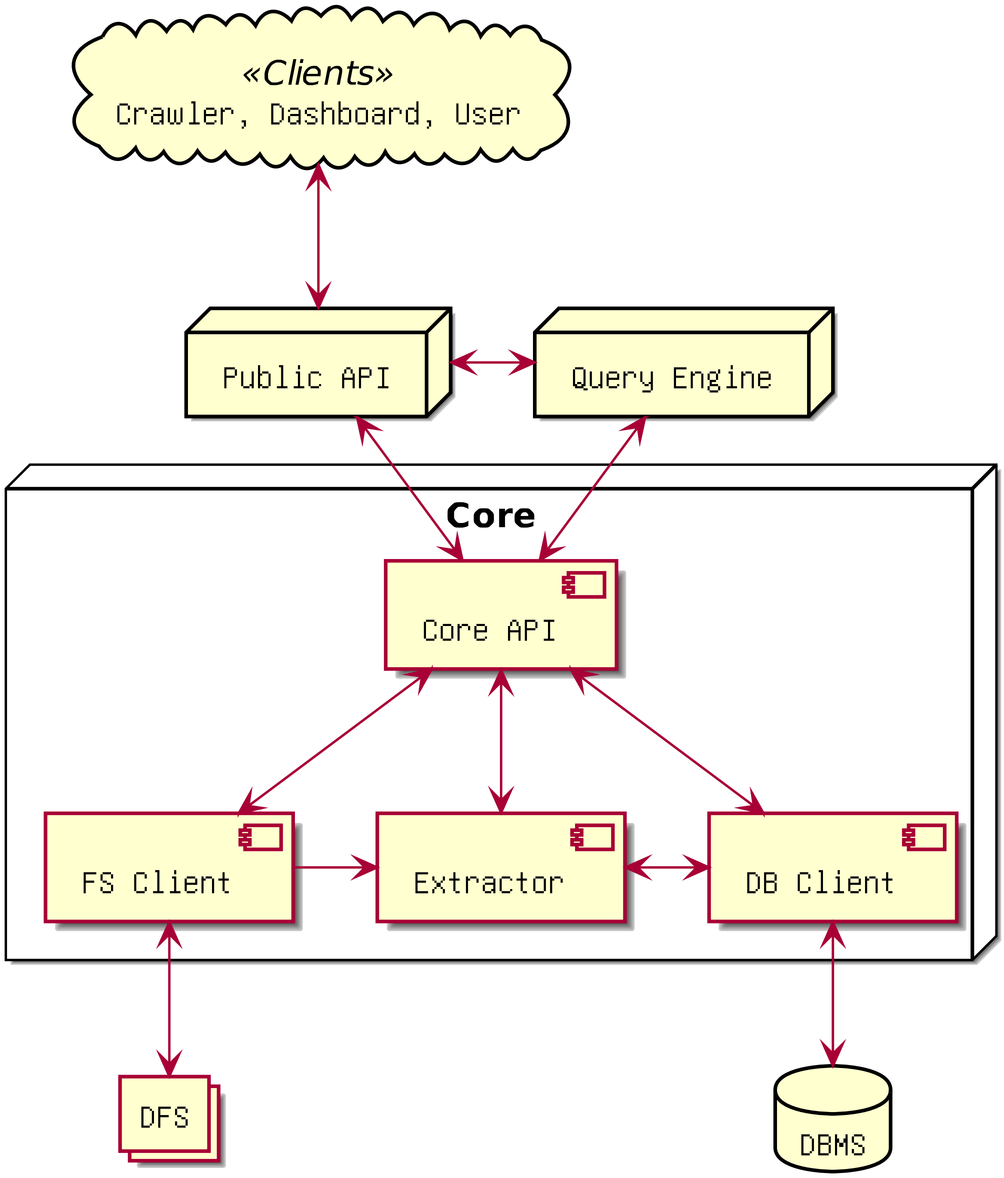
2.4.2 I/O Handling

2.4.3 Query Transformations

2.4.4 Concurrency

2.4.5 Logging

2.5 Quality Assurance


«Clients»
Crawler, Dashboard, User
Public API
Query Engine
Core
Core API
FS Client
Extractor
DB Client
DFS
DBMS

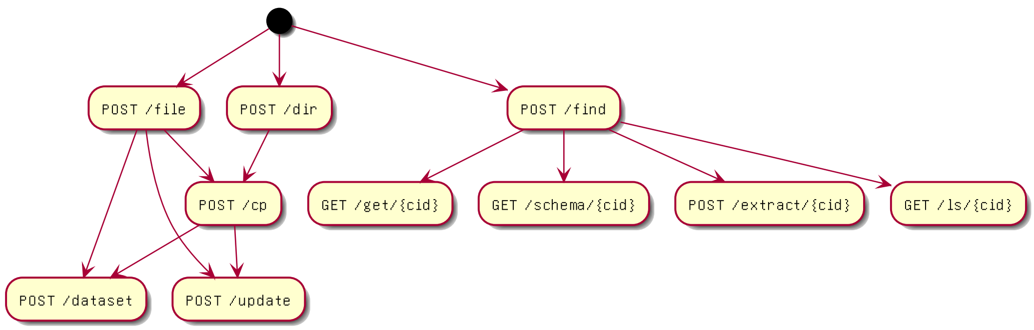


Figure 2.2: Core HTTP API endpoints in a common order of access

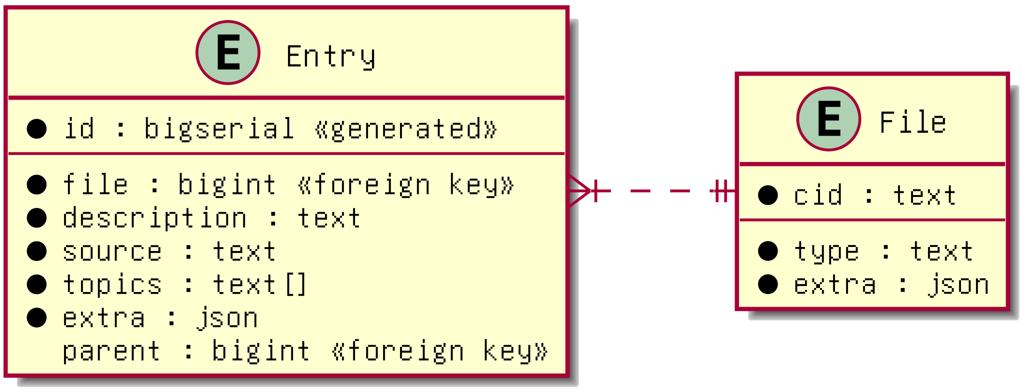


Figure 2.3: Database schema

3 Results and Discussion

3.1 Results

3.2 Discussion

4

Conclusion

A

Acknowledgment

B
Bibliography

[1] *Mission and Vision*. University of Science and Technology of Hanoi. `https://usth.edu.vn/en/abouts/Mission-et-Vision.html`.

[2] Huang Fang. "Managing Data Lakes in Big Data Era: What's a data lake and why has it became popular in data management ecosystem". *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems*, pp. 820–824. IEEE, 2015. doi:10.1109/CYBER.2015.7288049

[3] Chris Richardson. "1.4.1 Scale cube and microservices". *Microservice Patterns*. Manning Publications, 2018. ISBN 9781617294549.

[4] Brian Göetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer and David Holmes. "3.4. Immutability". *Java Concurrency in Practice*. Addison Wesley Professional, 2006. ISBN 9780321349606.

C Terms and Acronyms

C.1 Glossary

content a file or a directory. 2, 3, 5, 13

predicate an expression which is evaluated to either true or false. 3, 5

C.2 Acronyms

API application programming interface. 1, 2

CID content ID. 2, 3, 5

DB database. 3, 5

ID identifier. 2, 5, 13

USTH University of Science and Technology of Hanoi. 1, 2, 12