**Microservice and Spring Boot**

While implementing the project, we had to deal with multiple applications that can be written in different programming languages and use different data storage technologies. The Microservices as an architecture style can deploy application modules independently, perform different management operations on different modules, and generate small services for different modules. Each functional element can eventually become a functional unit that can be replaced and upgraded independently, and each small service communicates with each other through HTTP [1].

With Spring Boot, we can easily create standalone Spring projects that generate independent units of microservice functionality and integrate with mainstream frameworks. Meanwhile, Spring Boot can use the Spring project Boot page to build a project in seconds and export various forms of services such as REST APIs, WebSocket, Web, Streaming, Tasks and support for relational and non-relational databases.

In this project, we took these advantages of Spring Boot, to design and implement the application as shown in Figure 1. Overall, the application consists of 3 major services: post request to a fast-api service which allows python script to perform deep learning and machine learning algorithms, read intermediate result from Redis database and query and process metadata from MongoDB database. Following each service will be explored in depth.

**Python-Spring Boot (Java) Communication**

As mentioned in previous section, in this project, multiple services programmed by different languages need to be taken care. For instance, the Spring Boot is built on the Java platform as a microservice architecture, while the Python is known for its rich deep learning and analysis packages. To integrate the python script into Spring Boot project, although spring provides dependencies like Jython [2] that enables to embed Python code directly into Java code, the usage still is not friendly to native python developer. Therefore, we chose another popular approach for spring boot project, which wrap the Python code into a “dockerized” REST microservice and then communicate with spring application with HTTP.

In Figure 1, the data flow of transforming text queries to encodings as follow: First, the dockerized FAST-API solution is activated and available to take text queries request post from frontend web application. After the python script process the text queries, the intermediate result will be stored in Redis Database that allows the spring application to query and conduct further process. Under such development, the python machine learning algorithms can be developed independently so long as the expected IO can be accurately delivered in Redis database.

**Redis**

The Redis database is chosen as location to store intermediate encodings and cluster indices result from NLP deep learning algorithm and clustering algorithm coded in python script.

Redis is a key-value storage with in-memory operation system. Therefore, CPU is not the bottleneck of Redis, and it saves a lot of time on context-switching threads and doesn't have to worry about locks which can bring better maintainability, convenient development and debugging [3].

For this project, we chose a RedisJSON data structure to store in the Redis database. RedisJSON is a high-performance JSON document store that allows developers to build modern applications. It stores and processes JSON in memory to support response times of millions of operations per second at the sub-millisecond level. Native indexes, queries, and full-text searches for JSON documents allow developers to create secondary indexes to quickly query data [4].

In our application, the test result shows that the encoding transformation task for 100 queries can be completed in few milliseconds.

**MongoDB and MongoDB spark connector**

MongoDB is called the most developer-friendly database. Instead of emphasizing rows and columns in a traditional relational database, the entire table can be viewed as a Json document. MongoDB is also considered the Nosql database that most resembles a relational database in Nosql. Databases, collections, and Document objects like relational databases are retained.

The biggest feature of MongoDB is that the query language it supports is very powerful. Its syntax is similar to object-oriented query language, and it can almost achieve most of the functions similar to single table query of relational database, and also supports data index. The implementation of MongoDB in high availability and read/write load balancing is very simple and friendly.

These features of MongoDB are very convenient for high-performance data query. MongoDB supports Aggregate and MapReduce to process large-scale data analysis using the concept of divide and conquer. Spring Boot is very friendly to MongoDB. It is very convenient to use Spring Boot to process the query and operation of MongoDB. Spring Boot also provides component packages to support the use of MongoDB.

Spark's workflow can be summarized as 3 stages: create concurrent tasks, perform transformation operations on data such as Map, filter, Union, Intersect, etc., and then perform operations such as reduce, count, or simply collect results. Figure 4 shows a typical architecture for Spark and MongoDB deployments. Spark tasks are initiated by the Driver node of Spark and distributed through the Spark Master. For example, if we have four Spark Worker nodes, several executor computing processes on these nodes will start working at the same time. There is usually one executor for each core. Each executor obtains raw data from MongoDB independently, uses the analysis algorithm provided by Spark or uses a custom process to process the data, and writes the calculation result back to MongoDB. In this project, the interaction with MongoDB is done through the Mongo Spark connector (Figure 5) provided by MongoDB [3].

**Cosine Similarity Spark implementation**

Fortunately, in this project we only care about the query encoding similarity to each reference encoding instead of considering the cross similarity between them that stored in MongoDB. Thus, we can treat each cosine similarity computation as an isolated process without concerning about the communication between spark nodes.

Figure 6 shows the implementation code for computing cosine similarity on spark data frame queried from MongoDB. In this case, the partition key is the mongo DB hash key which is one to one projection to the image id under corresponding cluster id. The algorithms consider performing numerator and denominator before compute the final cosine similarity score.

**Spark Solution Analysis**

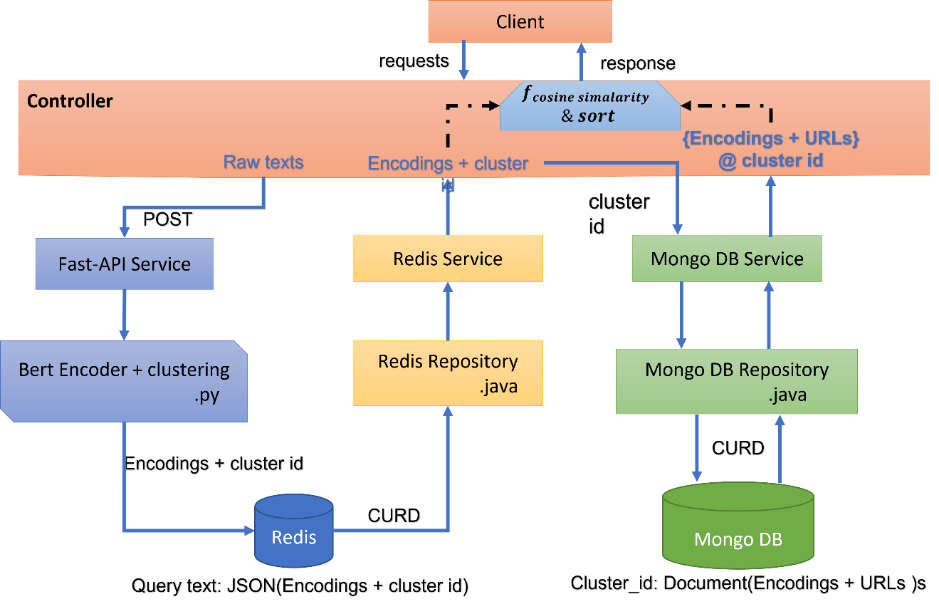


Figure Spring Boot Architecture Design

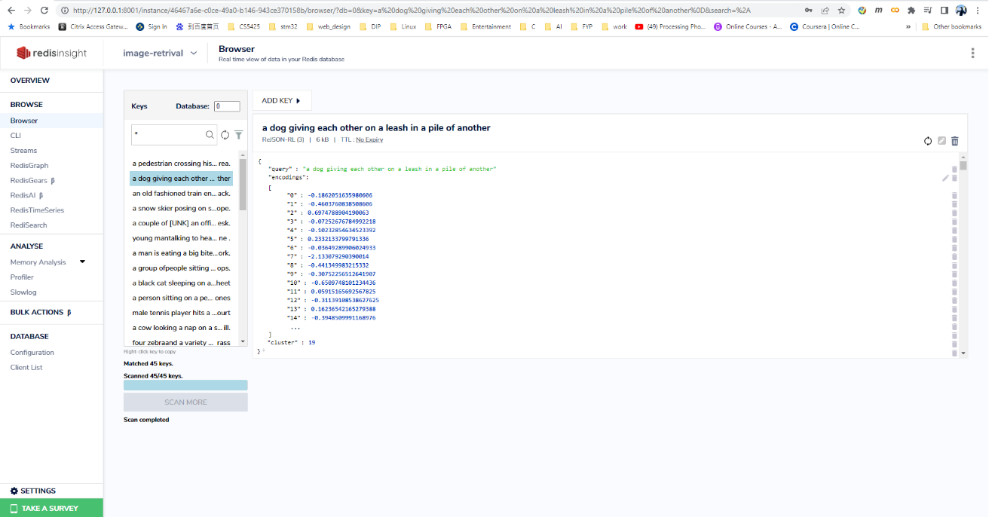


Figure Redis Data Structure

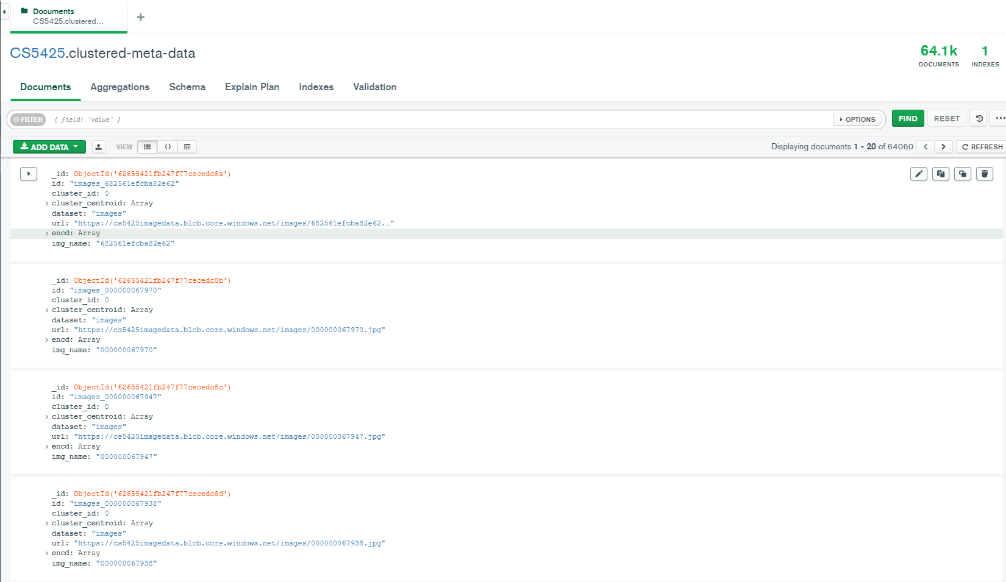


Figure MongoDB Data Structure

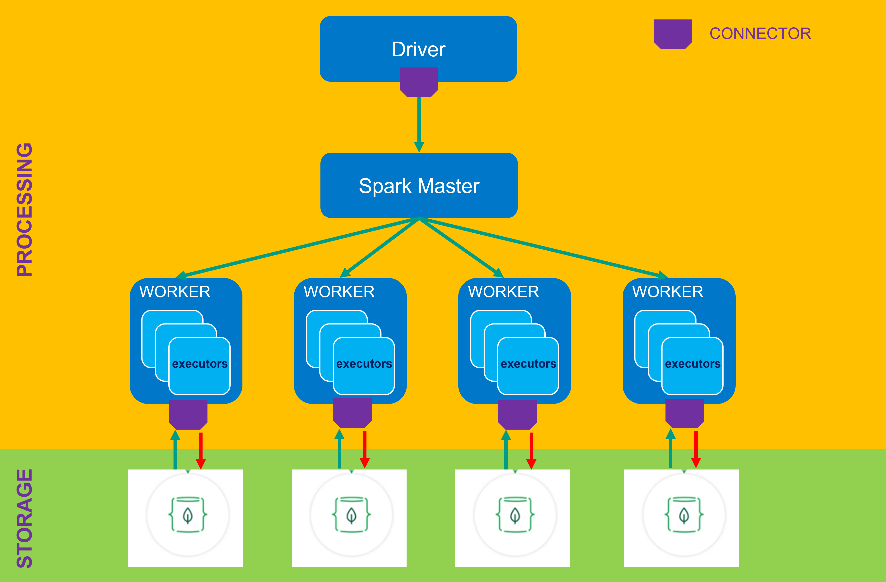


Figure : MongoDB Spark Architecture

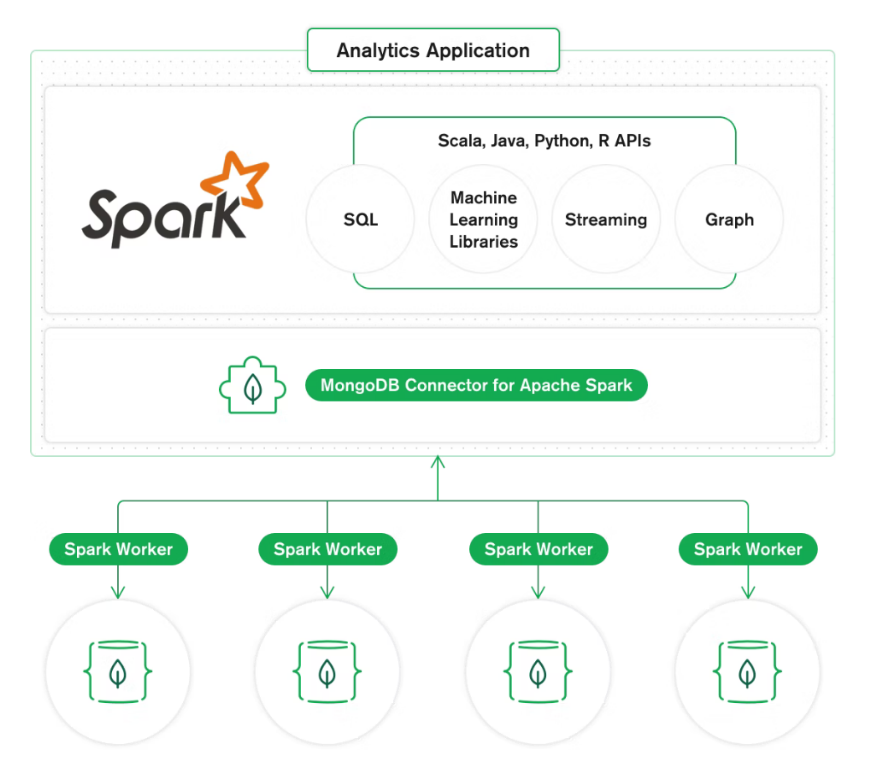


Figure : mongoDB spark connector



Figure Cosine Similarity Spark Implementation

# Bibliography

|  |  |
| --- | --- |
| [1] | J. Lewis and M. Fowler, "Microservices," 25 March 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html. [Accessed 16 April 2022]. |