



# Simple Search Engine Report

**Assignment 2: Simple Search Engine using Hadoop MapReduce, Cassandra, and other powerful tools**

**Student:** Davydov Danil

**Group:** DS-01

**Innopolis email address:** d.davydov@innopolis.university

**Telegram:** @tanelID

---

## 1. Introduction

- In this assignment I was asked to implement simple search engine using Spark, Cassandra, HDFS. I gained knowledge in doing it, also I implemented Map-Reduce pipelines
- 

## 2. Methodology

### 2.1 Design Choices

- **Data Pipeline:** The script `prepare_data.sh` and `prepare_data.py` are used to read parquet file, select 1000 texts and save them into `app/data` folder, in `.sh` script those files are copied to hdfs. If the user wants to index additional files he should paste `.txt` files into a new folder and pass the folder path into [index.sh](#) as an argument to index additional documents. The documents are concatenated to the files that were indexed and stored in from `app/data`, those additional files will also be placed in `/app/data`. Next I have 2 Map-Reduce pipelines to store files in cassandra, during this pipelines all metrics for BM25 calculation will be determined and stored in cassandra. The last stage is searching with [search.sh](#) and [query.py](#). With this scripts I calculate scores for each document by the query and print top-10 the most relevant documents. The output has filename, score, and part of a text.

The pipeline looks as follows:

parquet file ---> prepare\_data scripts ---> [index.sh](#) ---> 2 Map-Reduce pipelines ---> search for relevant

^

|

folder with additional files to index (optional)

- **Indexing Strategy:** My cassandra schema is defined by script:

```
CREATE KEYSPACE IF NOT EXISTS document_index
WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'};

USE document_index;

CREATE TABLE IF NOT EXISTS vocabulary (
    term text PRIMARY KEY
);

CREATE TABLE IF NOT EXISTS document_lengths (
    doc_id text PRIMARY KEY,
    doc_length int
);

CREATE TABLE IF NOT EXISTS term_frequencies (
    term text,
    doc_id text,
    frequency int,
    PRIMARY KEY (term, doc_id)
);

CREATE TABLE IF NOT EXISTS document_frequencies (
    term text PRIMARY KEY,
    doc_frequency int
);

CREATE TABLE IF NOT EXISTS document_texts (
    doc_id text PRIMARY KEY,
    content text
);
```

The table `document_texts` is created to show slice of a text in output, other tables are needed to calculate BM25 score for documents by query. The initialization of schema is in [app.sh](#) script.

- **Ranking Model:** The bm25 has 2 parameters which I defined as  $k_1 = 1.2$  and  $b = 0.75$

## 2.2 Implementation Details

### Indexer (MapReduce Pipeline)

- I splitted the code into 2 MapReduce pipelines the first calculates all metrics while the second loads them into cassandra, such that my code is simpler to understand.

### Ranker (Spark RDD)

- **Query Processing:** I used the same strategy to preprocess the texts and query, I remove stopwords and stem the tokens.
- **BM25 Calculation:** I use a modification of BM25 to rank documents (Variant Lucene/Elasticsearch), the calculation looks like

### BM25 Scoring Formula (Lucene/Elasticsearch Variant)

#### IDF (Inverse Document Frequency):

$$IDF(t) = \log \left( \frac{N - df(t) + 0.5}{df(t) + 0.5} + 1 \right)$$

#### TF (Term Frequency):

$$TF(t, d) = \frac{tf(t, d) \cdot (k_1 + 1)}{tf(t, d) + k_1 \cdot \left( 1 - b + b \cdot \frac{|d|}{avgdl} \right)}$$

#### Full BM25 Score:

$$\text{Score}(q, d) = \sum_{t \in q} IDF(t) \cdot TF(t, d)$$

Here is the comparison of classic BM25 and Variant Lucene/Elasticsearch:

## BM25 vs. Lucene/Elasticsearch BM25 Variants: IDF and Score Ranges

Feature	Classic BM25	Lucene/Elasticsearch BM25
<b>IDF for Common Terms</b>	Can go negative for very common terms	Always non-negative due to the "1+" term
<b>IDF for Terms in All Docs</b>	Negative values possible	Floors at 0 to avoid negative contribution
<b>Score Range - Lower Bound</b>	Can produce negative scores for common terms	Always non-negative scores
<b>Impact on Ranking</b>	Higher contrast between common and rare terms	More balanced influence between common and rare terms

- **Top-K Retrieval:** Spark operations to rank and filter results.

I use spark RDD for calculating BM25 score in parallel in [query.py](#):

```
# other code
doc_ids_rdd = sc.parallelize(list(doc_lengths.keys()))
# other code
```

```

# other code
doc_scores = doc_ids_rdd.map(
    lambda doc_id: (
        doc_id,
        calculate_bm25(
            query_terms,
            doc_id,
            doc_lengths.get(doc_id, 0),
            avg_doc_len,
            term_freqs,
            doc_freqs,
            total_docs
        )
    )
)

top_docs = doc_scores.sortBy(lambda x: -x[1]).take(10)
# other code

```

## 3. Demonstration

### 3.1 How to Run the Project

Run

```

docker-compose build
docker-compose up

```

If you want to change query: change it directly in the end of [app.sh](#):

```

bash search.sh "paste_your_query"

```

If you want to index additional documents add path to the folder with additional txt files:

```

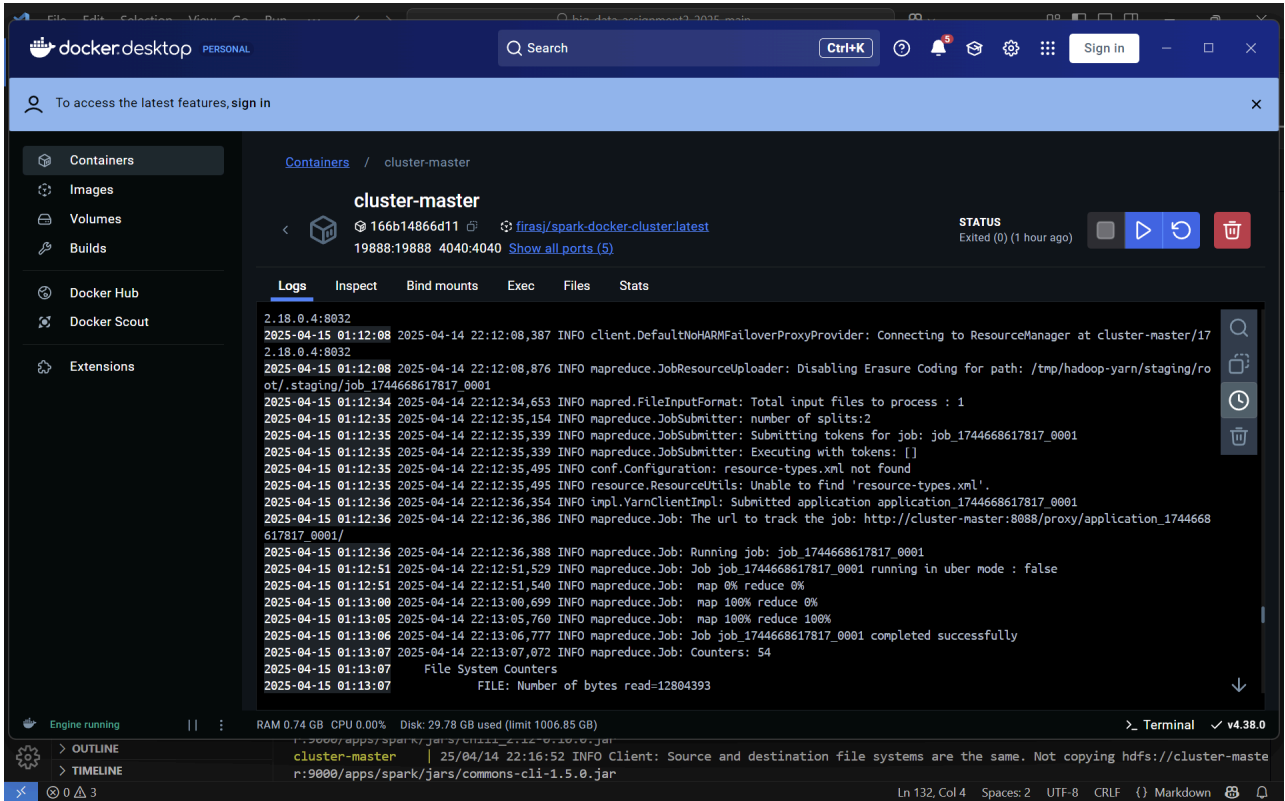
bash index.sh /app/your_folder

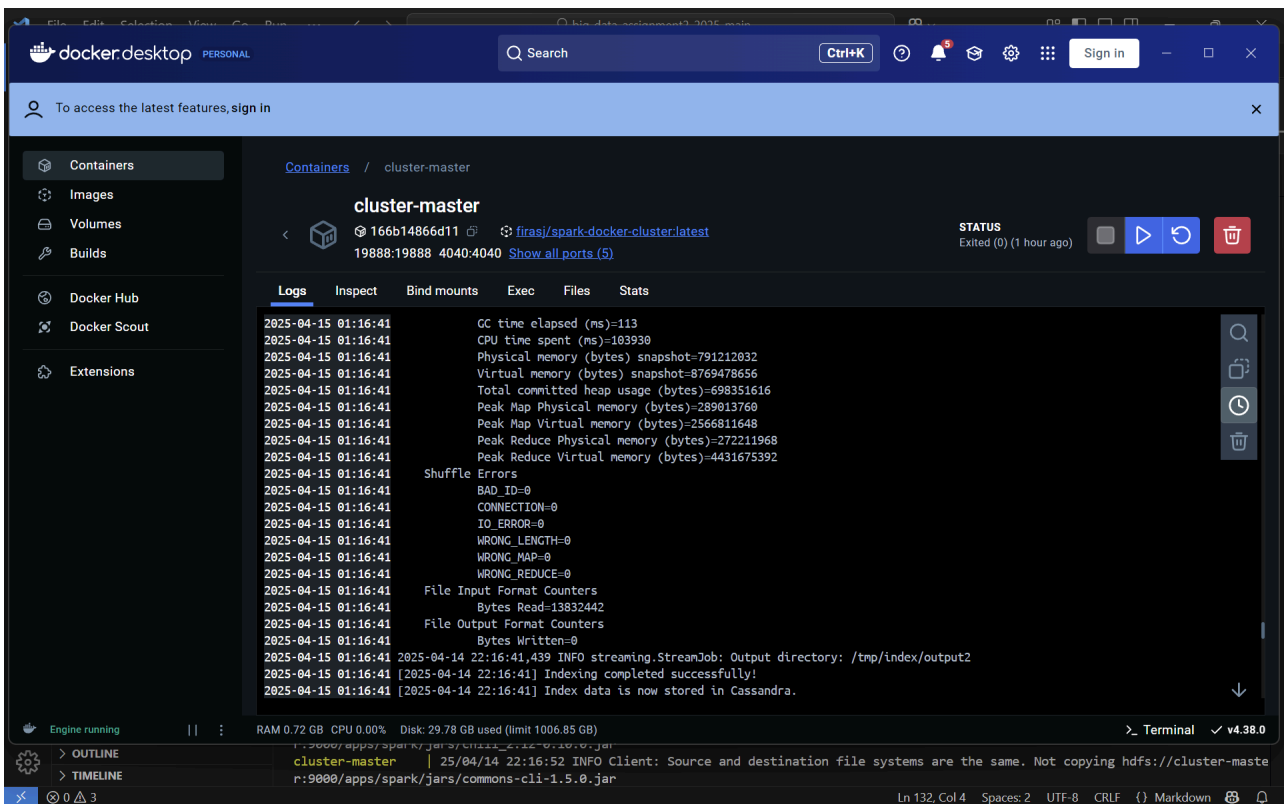
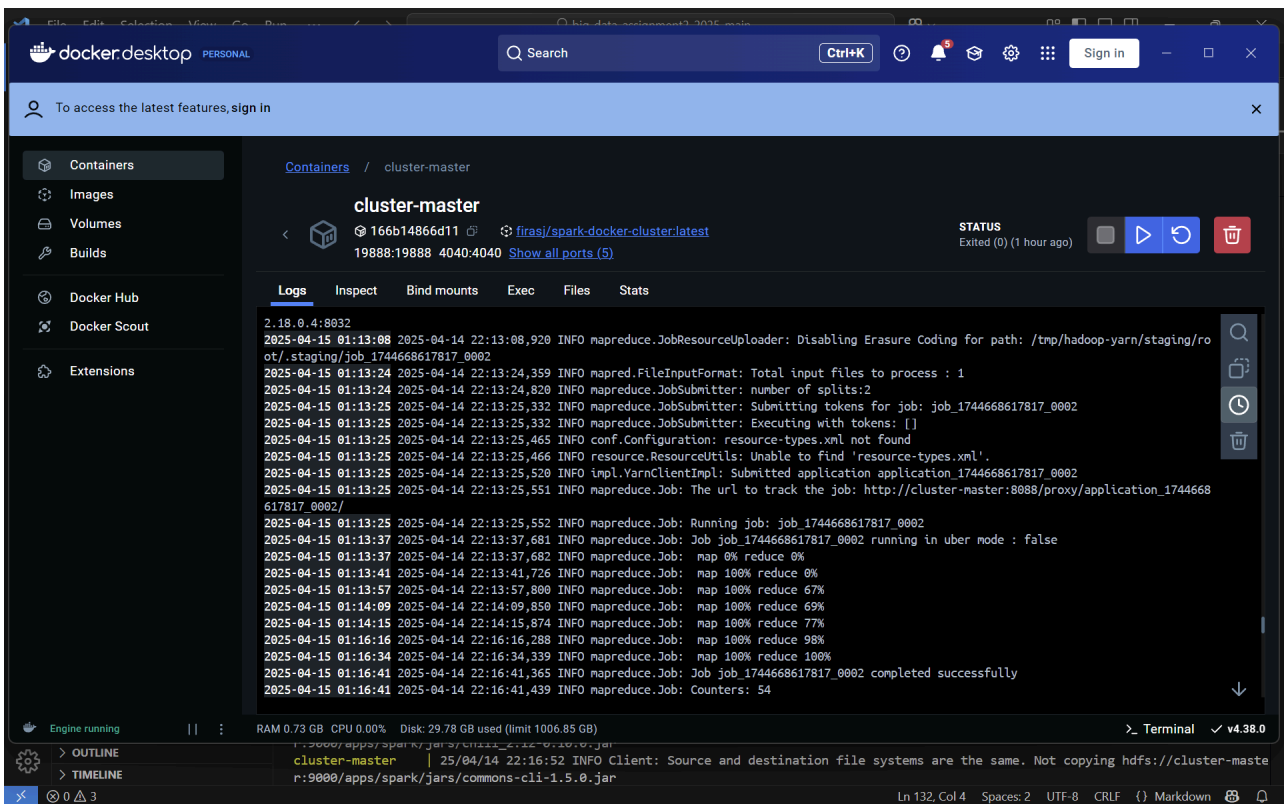
```

## 3.2 Screenshots & Results

### Indexing Success

- **Screenshot 1:** Fullscreen terminal output showing successful indexing of documents.





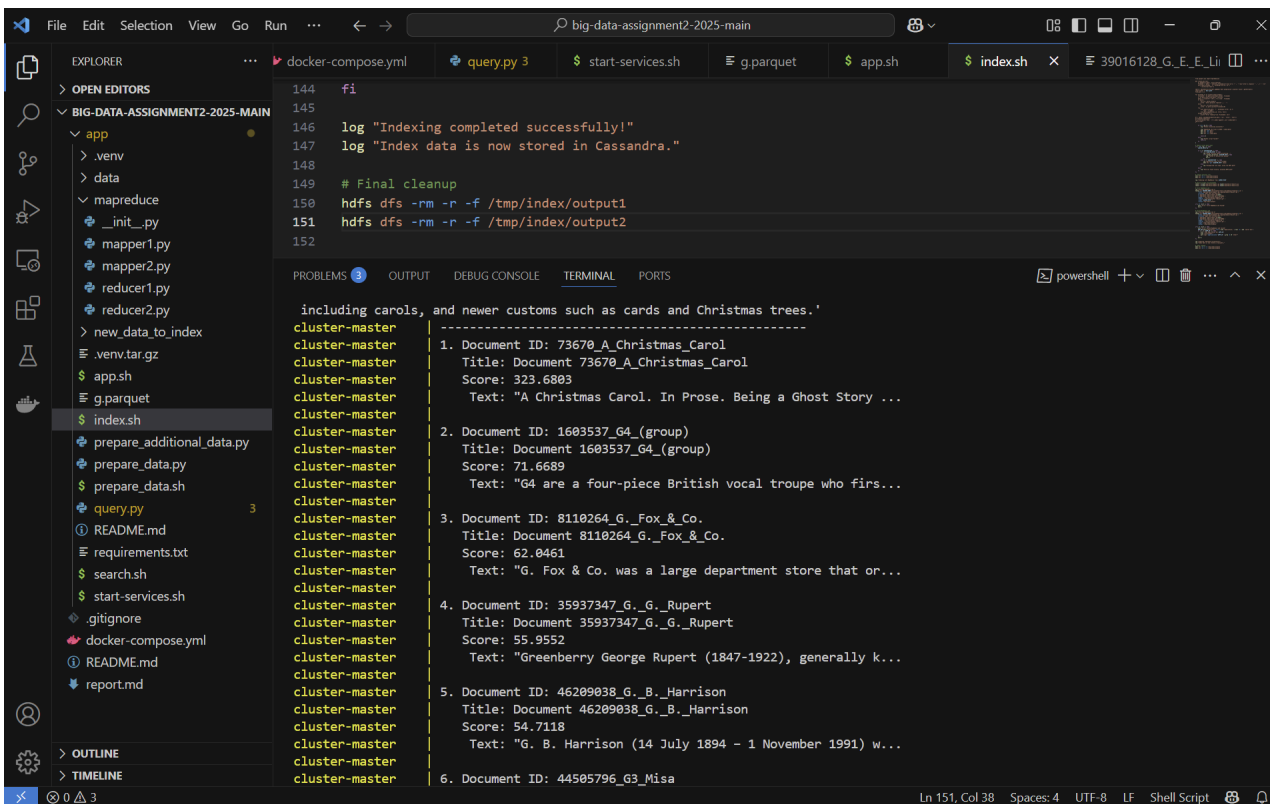
## Query Execution

- Query 1: "Famous chemistry scientists"





story of Ebenezer Scrooge, an elderly miser who is visited by the ghost of his former business partner Jacob Marley and the spirits of Christmas Past, Present and Yet to Come. After their visits, Scrooge is transformed into a kinder, gentler man. Dickens wrote A Christmas Carol during a period when the British were exploring and re-evaluating past Christmas traditions, including carols, and newer customs such as cards and Christmas trees."



The screenshot shows a VS Code editor with a project named 'big-data-assignment2-2025-main'. The Explorer panel on the left shows a file tree with folders like 'app', 'data', 'mapreduce', and 'new\_data\_to\_index'. The 'app' folder contains files like '.\_env', 'data', 'mapreduce', '.\_init\_.py', 'mapper1.py', 'mapper2.py', 'reducer1.py', 'reducer2.py', 'new\_data\_to\_index', '.\_env.tar.gz', '.\_app.sh', '.\_g.parquet', '.\_index.sh', 'prepare\_additional\_data.py', 'prepare\_data.py', 'prepare\_data.sh', 'query.py', 'README.md', 'requirements.txt', 'search.sh', 'start-services.sh', '.\_gitignore', '.\_docker-compose.yml', '.\_README.md', and '.\_report.md'. The 'new\_data\_to\_index' folder is selected. The main editor shows a file named 'query.py' with the following content:

```
144 fi
145
146 log "Indexing completed successfully!"
147 log "Index data is now stored in Cassandra."
148
149 # Final cleanup
150 hdfs dfs -rm -r -f /tmp/index/output1
151 hdfs dfs -rm -r -f /tmp/index/output2
152
```

The terminal panel at the bottom shows the output of the 'index.sh' script, which is a query result. The output is as follows:

```
including carols, and newer customs such as cards and Christmas trees.'
-----
1. Document ID: 73670_A_Christmas_Carol
   Title: Document 73670_A_Christmas_Carol
   Score: 323.6803
   Text: "A Christmas Carol. In Prose. Being a Ghost Story ...

2. Document ID: 1603537_G4_(group)
   Title: Document 1603537_G4_(group)
   Score: 71.6689
   Text: "G4 are a four-piece British vocal troupe who firs...

3. Document ID: 8110264_G_Fox_&_Co.
   Title: Document 8110264_G_Fox_&_Co.
   Score: 62.0461
   Text: "G. Fox & Co. was a large department store that or...

4. Document ID: 35937347_G._G._Rupert
   Title: Document 35937347_G._G._Rupert
   Score: 55.9552
   Text: "Greenberry George Rupert (1847-1922), generally k...

5. Document ID: 46209038_G._B._Harrison
   Title: Document 46209038_G._B._Harrison
   Score: 54.7118
   Text: "G. B. Harrison (14 July 1894 - 1 November 1991) w...

6. Document ID: 44505796_G3_Misa
```

Caption: this query is used with additional documents that were indexed, the folder is app/new\_data\_to\_index  
this path was used as the argument to [index.sh](#) script. This query is a demonstration that my code can index additional documents