Simple Search Engine using Hadoop MapReduce

1. Methodology

This search engine was built as a modular, distributed pipeline designed for indexing and querying large-scale documents using the BM25 ranking function. The implementation leverages Hadoop MapReduce for document preprocessing, Apache Cassandra for distributed storage, and Apache Spark for ranking search results.

1.1 Indexing Pipeline

MapReduce

I utilize a two-step MapReduce process:

- mapper1.py:
  - Parses each input line into doc_id, doc_title, and doc_text.
  - Tokenizes and normalizes text by removing punctuation and converting to lowercase.
  - Computes term frequencies (TF) per document.
  - Stores document metadata and term frequencies into Cassandra (docs and tfs tables).
  - Emits intermediate output in the format <term>\t<doc_id> for computing document frequencies.
- reducer1.py:
  - Aggregates the number of documents in which each term appears (document frequency, DF).
  - Stores DF values into the dfs table in Cassandra.

Index Creation Script (index.sh)
This script:

- Runs the Hadoop MapReduce job using the Hadoop streaming API.
- Uses a compressed Python virtual environment (.venv.tar.gz) for dependency isolation.
- After indexing, executes app.py to compute the average document length (dl_avg) across the corpus, storing it in a dedicated Cassandra table.

1.2 Cassandra Data Storage

The following Cassandra tables are used:

- docs(doc_id TEXT PRIMARY KEY, doc_title TEXT, dl INT): Stores titles and document lengths.
- tfs(doc_id TEXT, term TEXT, tf INT, PRIMARY KEY (doc_id, term)): Stores term frequencies.
- dfs(term TEXT PRIMARY KEY, df INT): Stores document frequencies.
- dl_avg(index_name TEXT PRIMARY KEY, dl_avg DOUBLE): Stores average document length.

1.3 Query Processing and Ranking

Queries are processed using query.py, which uses Apache Spark to:

- Load the list of doc_ids from Cassandra.
- Preprocess the query (tokenization, normalization).
- For each document, compute a BM25 score

1.4 Script Automation

- app.sh: Executes the indexing and retrieval pipelines.
- search.sh: Accepts a query string and triggers query.py.

1. Demonstration

How to run

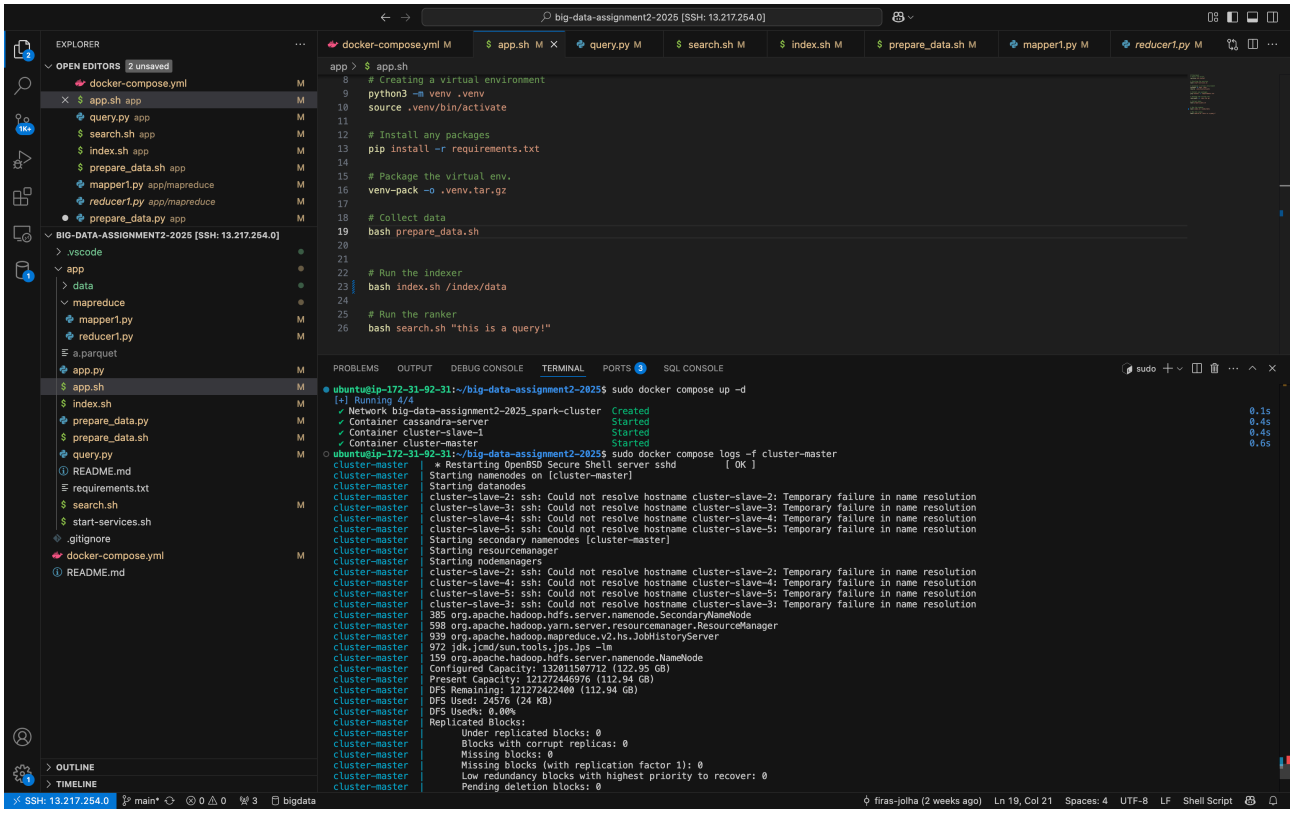
Step 1: Install prerequisites:

Docker

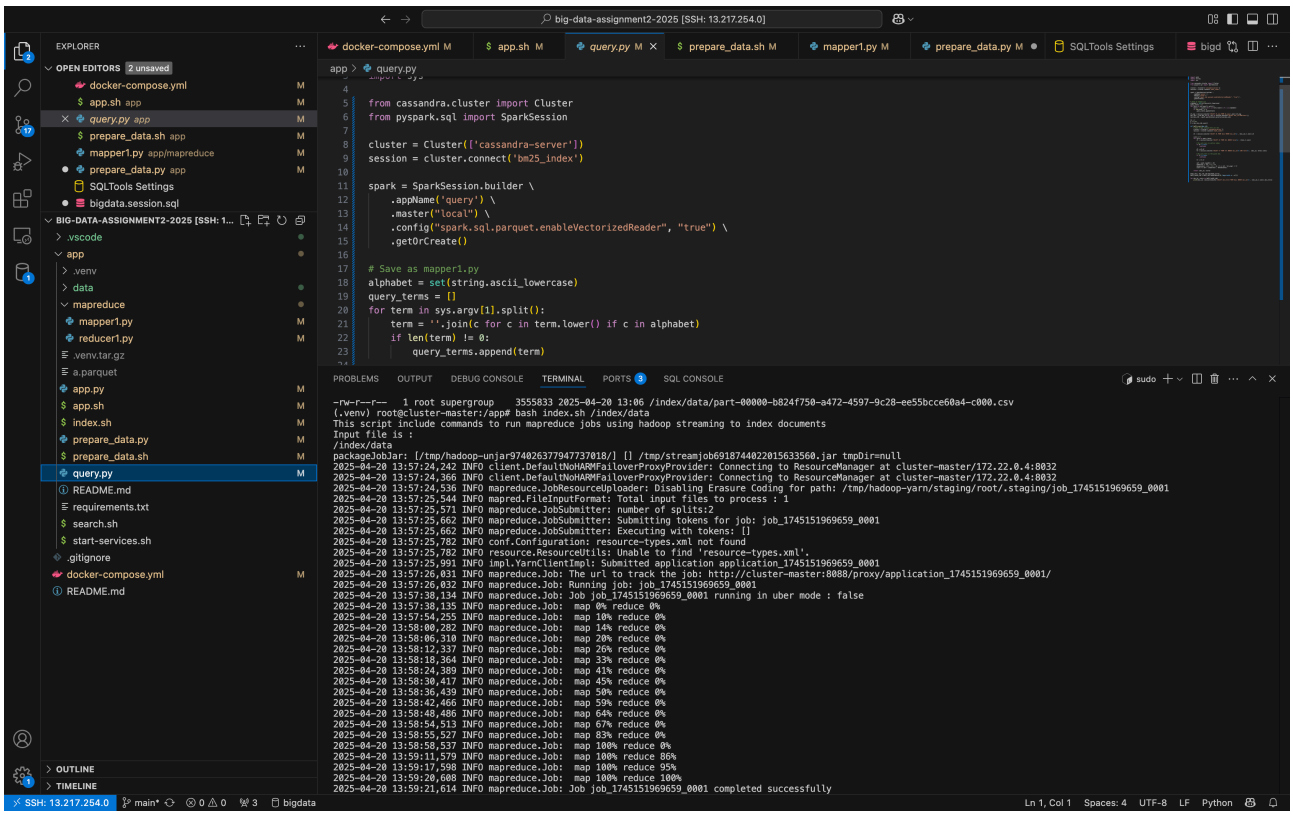
Docker compose

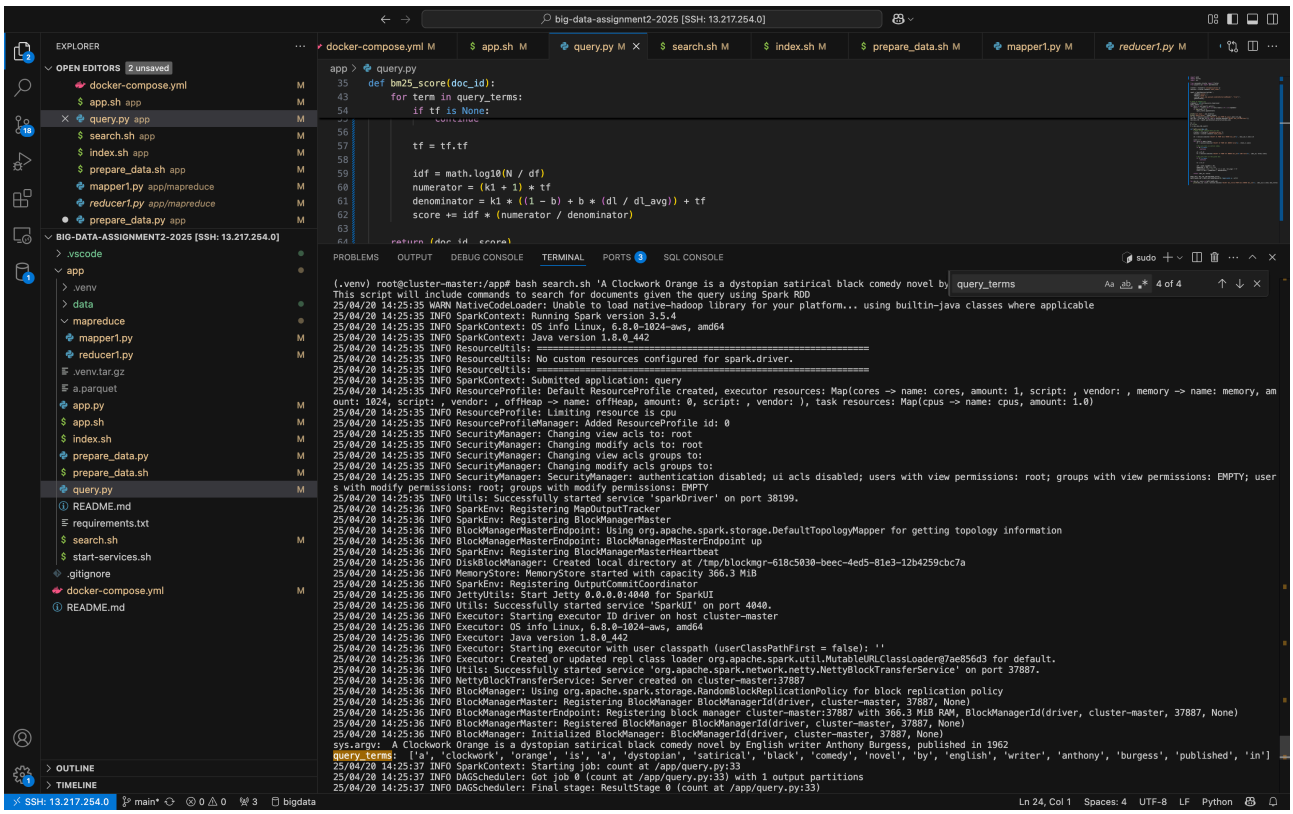
Step 2: Run the command: docker compose up

This will create 3 containers, a master node and a worker node for Hadoop, and Cassandra server. The master node will run the script app/app.sh as an entry point.



Here’s an example of indexer work:





And we see the output (top-10). I entered the first sentence of a 843 document as a query: