CSCI 5408

DATA MANAGEMENT AND WAREHOUSING

LAB - 5

Banner ID: B00958098
GitLab Assignment Link:
https://git.cs.dal.ca/abhishekk/csci5408_w24_b00958098_abhishek_kapoor

Table of Contents

Deliverable #1 ..... **3**
Direction 3
Deliverable #2 ..... **6**
Direction 6
Deliverable #3 ..... **7**
Direction 7
Deliverable #4 ..... **9**
Direction 9
Deliverable #5 ..... **9**
Direction 9

Deliverable #1

Direction

Screenshots of the step-by-step process followed to create the Apache Spark (GCP Dataproc) cluster and execute the job (WordCounter.jar) file on it.

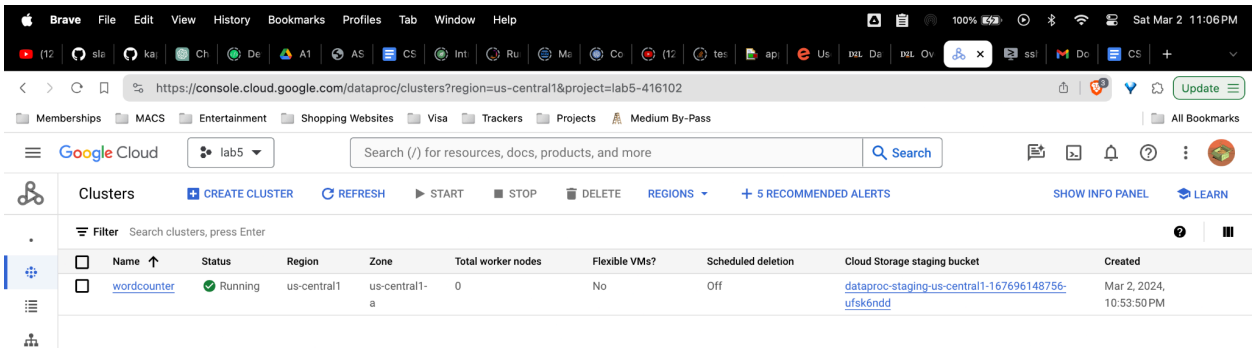

Fig. 1 WordCounter cluster being created

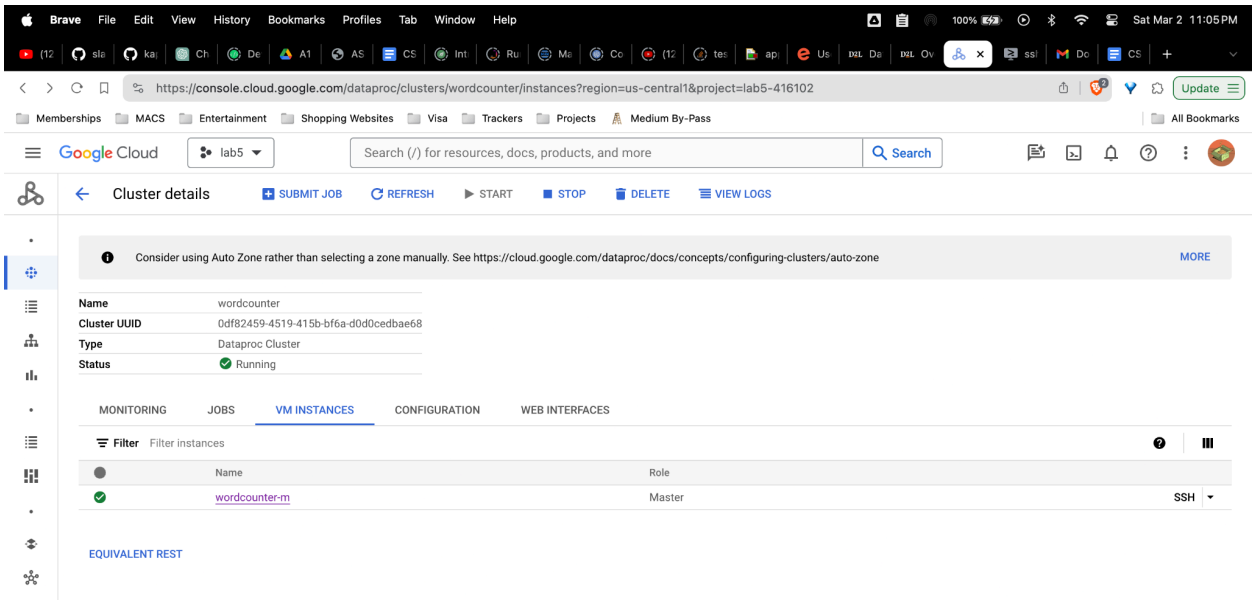

Fig. 2 Cluster Dashboard

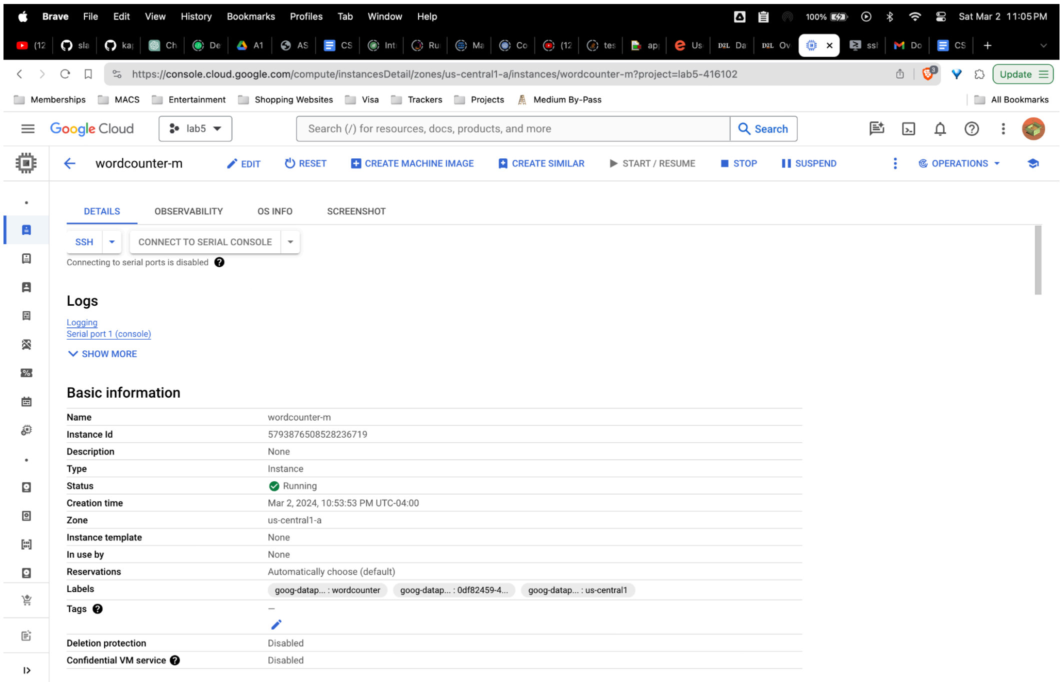

Fig. 3 Wordcounter VM instances

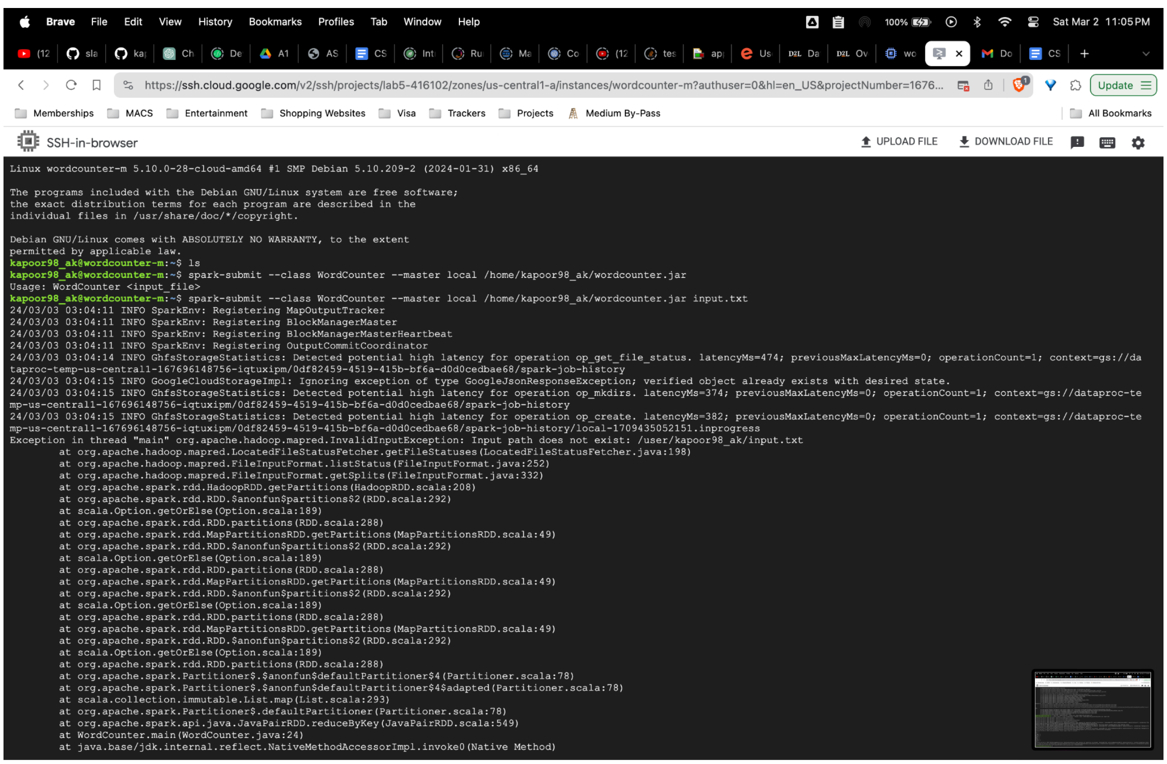

Fig. 4 Spark-submit command being run with appropriate flags to run the jar

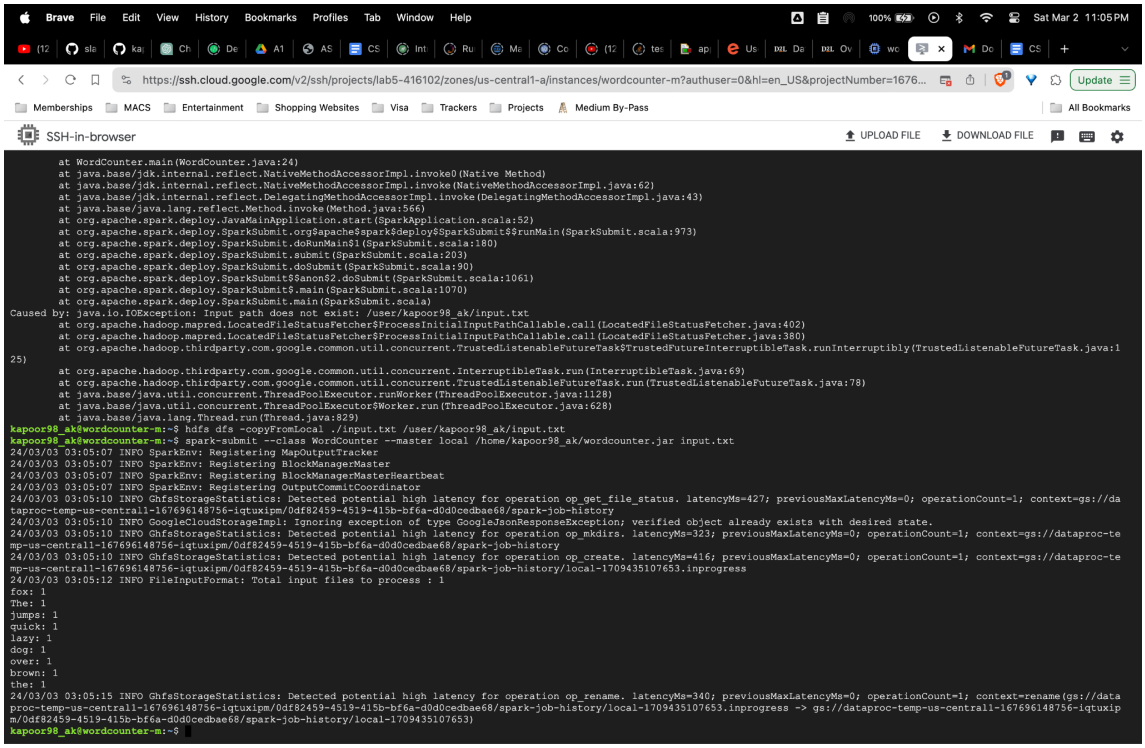

Fig. 5 Expected Results shown in the output

Deliverable #2

Direction

Report any challenges faced while executing the .jar file on the Apache spark cluster.

During the lab, I faced these challenges:

1. **Configuration and Setup:**
    - Setting up the Spark cluster environment correctly was a significant challenge as I could not follow properly in the Lab and then could not follow the Lab 5 PDF.
    - It involved taking help from other classmates, and ensuring that all the servers in the cluster have the required dependencies and configurations.
    - To not repeat this in future, I would test the setup on a small scale before deploying to a larger cluster.

2. **Resource Management and Optimization:**
    - Efficiently managing and optimizing resources on a Spark cluster is crucial for performance and cost management.
    - Issues can be resolved by fine-tuning the Spark configurations for memory, CPU, and other resource optimization.
    - I utilized Spark's built-in monitoring to monitor resource utilization.

3. **Debugging and Logging:**
    - Debugging issues on a distributed Spark cluster was challenging due to the distributed nature of the computation. Identifying and resolving errors, especially in a production environment, could be very time-consuming without proper logging and debugging mechanisms.
    - Enabling logging can help us to monitor job progress and identify errors. Incorporate proper error handling and logging within your application code to aid in troubleshooting.

Deliverable #3

Direction

Explanation of the Java Spark program with the screenshots of the code.

Main Class - WordCounter:

1. This class serves as the entry point for the application.
2. It checks if the input file path is provided as a command-line argument. If not, it prints an error message and exits.
3. If the input file path is provided, it creates an object of WordCounterProcessor and calls the process method to handle the word counting.

Processor Class - WordCounterProcessor:

1. This class has the processing logic for word counting using Apache Spark.
2. Methods:
    a. **process()**:
        - Takes the input file path as a parameter.
        - Initializes Spark configuration and context using helper methods.
        - Read the input file, split lines into words, map words to tuples, reduce word counts, collect results, print results, and finally stop the Spark context.
3. Helper Methods:
    a. **createSparkConf()**:
        - Creates a Spark configuration with the application name set to "WordCounter" and the master set to "local."
    b. **readInputFile()**:
        - Reads the input file using the provided Spark context and returns a JavaRDD<String> containing lines from the file.
    c. **splitLinesIntoWords():**
        - Takes a JavaRDD<String> of lines, splits each line into words, and returns a new JavaRDD<String> containing individual words.
    d. **mapWordsToTuples()**:
        - Takes a JavaRDD<String> of words and maps each word to a tuple (word, 1) using mapToPair, resulting in a JavaPairRDD<String, Integer>.
    e. **reduceWordCountsByKey()**:
        - Takes a JavaPairRDD<String, Integer> and reduces the tuples by key (word) by summing up the values, resulting in a new JavaPairRDD<String, Integer>.
    f. **collectResults()**:
        - Takes the reduced word counts as a JavaPairRDD<String, Integer> and collects the results as a list of tuples.
    g. **printResults()**:
        - Takes the list of tuples and prints each word and its count to the console.

This approach follows the SOLID principles by separating concerns and ensuring that each method has a single responsibility. It improves code readability by using meaningful names. The Spark-specific logic is encapsulated within the WordCounterProcessor class.

Deliverable #4

Direction

Provide references of any sources used for this lab.

References provided in the report.

Deliverable #5

Direction

Submit code on Gitlab.

GitLab Repository:
https://git.cs.dal.ca/abhishekk/csci5408_w24_b00958098_abhishek_kapoor