**EXP 2: Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.**

**AIM:**

To run a basic Word Count MapReduce program.

**Procedure:**

Ensure that Hadoop is properly installed and configured on WSL. If it’s not installed, follow the [Hadoop installation guide for WSL](https://docs.microsoft.com/en-us/windows/wsl/install).

**2. Write the Word Count Program**

Create a Java file for the Word Count program. Here’s a simple example:

**WordCount.java**

java

Copy code

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {

private final static IntWritable one = new IntWritable(1);

private Text word = new Text();

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String[] words = value.toString().split("\\s+");

for (String wordStr : words) {

word.set(wordStr);

context.write(word, one);

}

}

}

public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

result.set(sum);

context.write(key, result);

}

}

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "word count");

job.setJarByClass(WordCount.class);

job.setMapperClass(TokenizerMapper.class);

job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

**3. Compile the Word Count Program**

Compile the WordCount.java program using Hadoop’s classpath:

bash

Copy code

javac -classpath $(hadoop classpath) -d WordCount\_classes WordCount.java

Create a JAR file from the compiled classes:

bash

Copy code

jar cf wordcount.jar -C WordCount\_classes/ .

**4. Run Hadoop Word Count**

Upload your input file to HDFS:

bash

Copy code

hdfs dfs -mkdir /input

hdfs dfs -put /path/to/your/inputfile.txt /input

Run the Hadoop job:

bash

Copy code

hadoop jar wordcount.jar WordCount /input /output

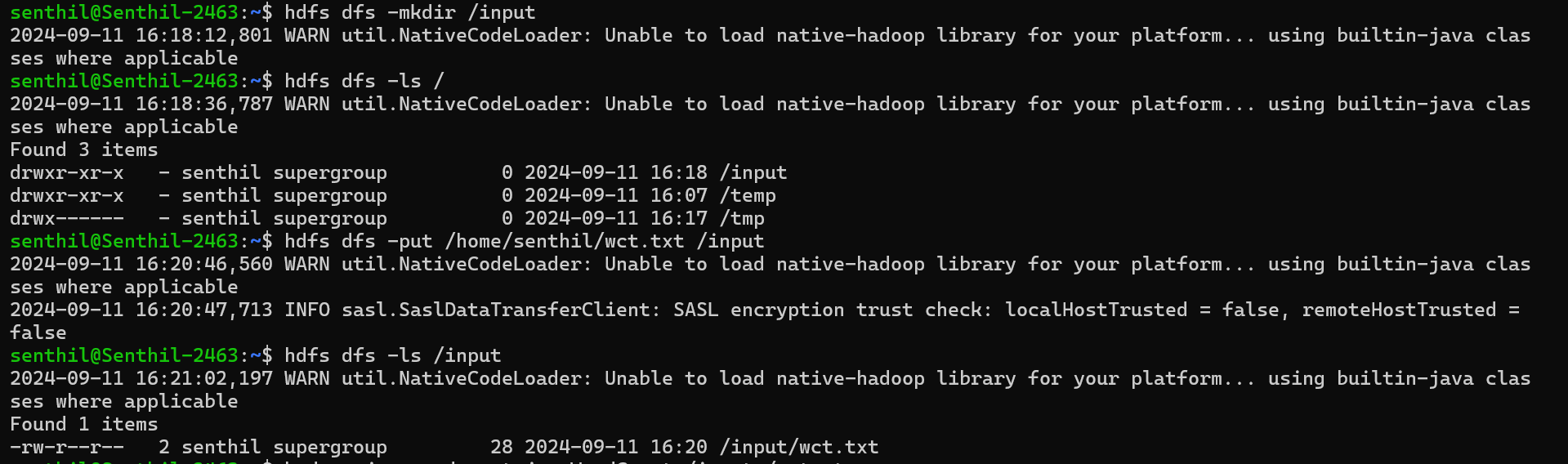
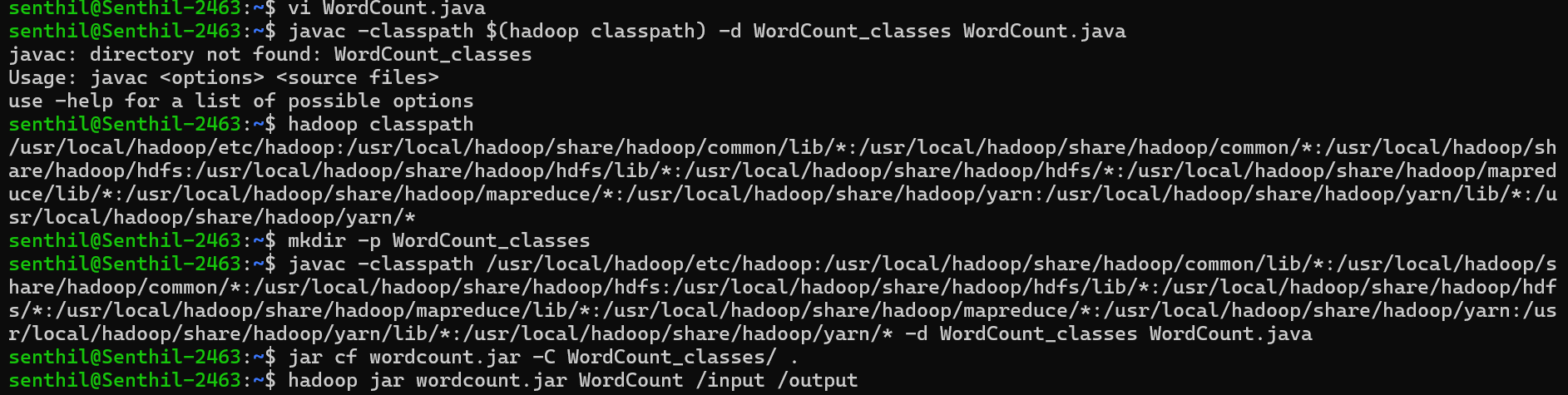
**5. Check the Output**

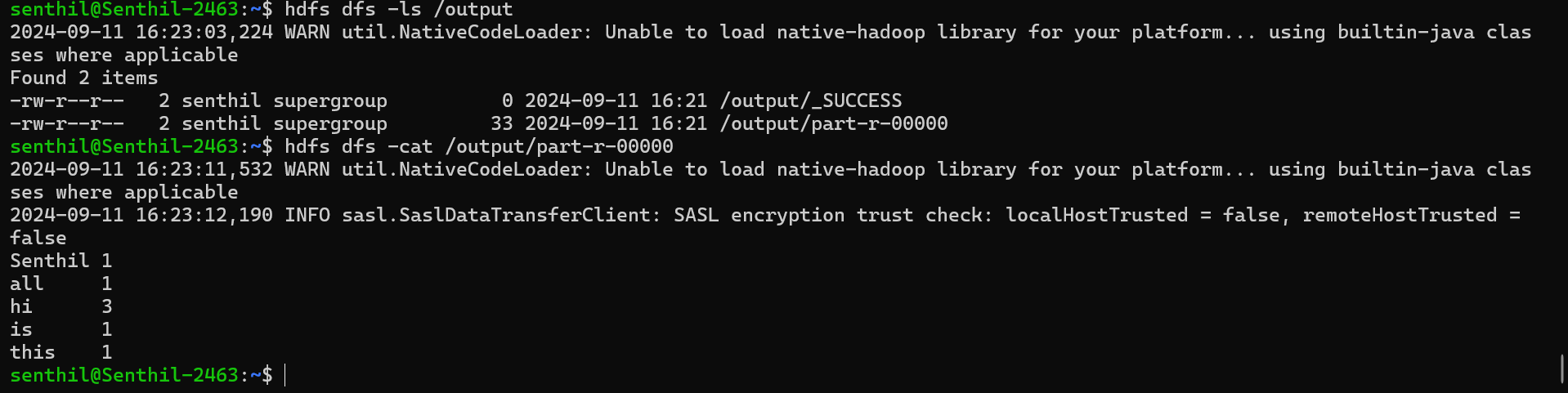
After the job completes, check the output:

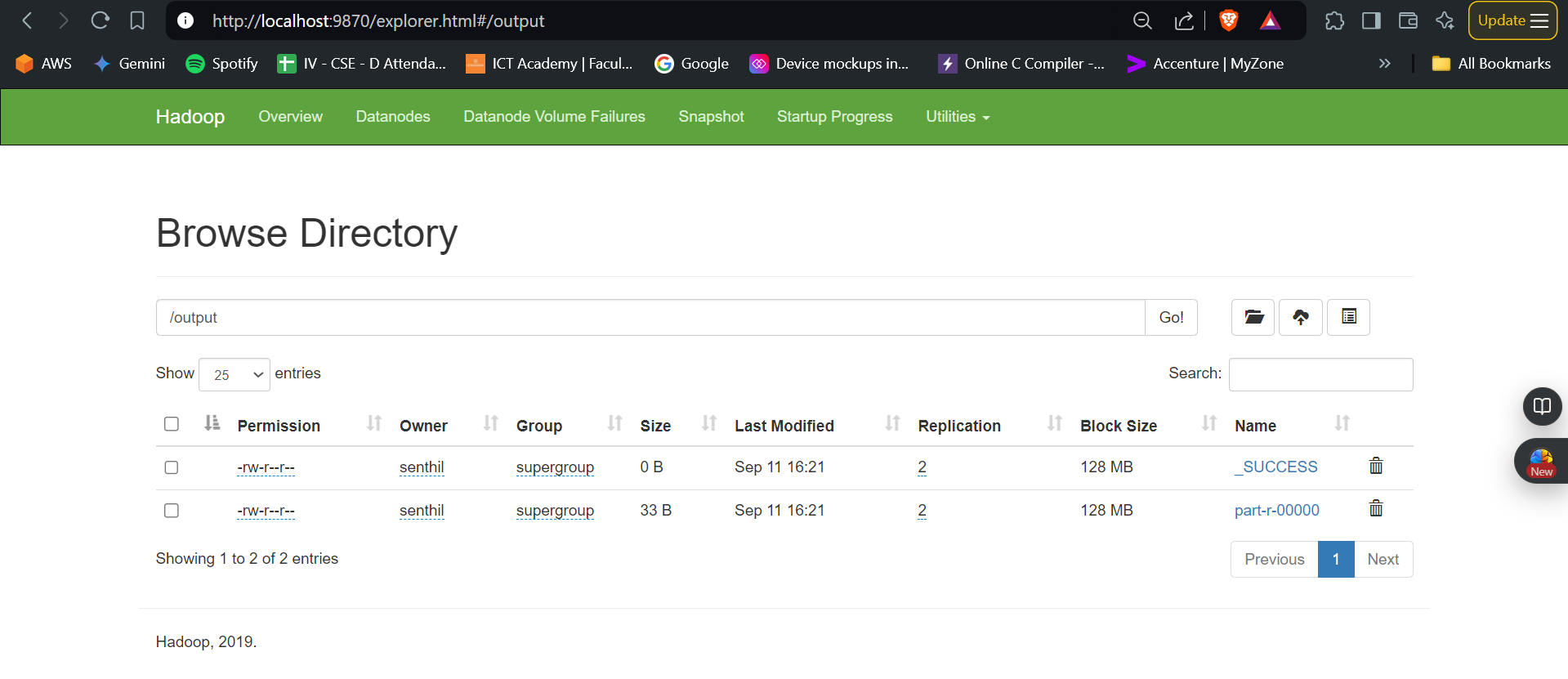
bash

Copy code

hdfs dfs -cat /output/part-r-00000







https://chatgpt.com/c/66e11ec5-56d8-800d-bd81-02c05c2e836e