**University of Wolverhampton**

**School of Mathematics and Computer Science**

**5CS022 Distribute and Cloud Systems Programming**

**Workshop 4: The Apache Spark Framework**

# Overview

Apache Spark is an open-source data processing framework which can perform analytic operations on Big Data in a distributed environment. It is compatible with both the Scala and Java programming languages.

This workshop shows you how to set up a simple Spark project in Eclipse and run a word counting program in Java.

## 1. Configuring Hadoop

To enable Apache Spark to work with Eclipse, some Hadoop utilities need to be available for Spark to call. Rather than downloading and installing the whole of Hadoop, which is about 500MB, on Canvas, there is a file called "hadoop.zip" which contains just the utilities that Spark needs for this workshop. Download it and extract the contents so that they are in the directory **C:\hadoop\bin**

**OR,**

You can clone this repo: <https://github.com/steveloughran/winutils/> and copy the contents of Hadoop-3.0.0 to the C drive and rename the folder to **hadoop.**

If you want to clone, you need to have [Git](https://git-scm.com/downloads) installed on your laptop.

Once installation is done, simply open either Git Bash or your command terminal and type:

*git clone* [*https://github.com/steveloughran/winutils.git*](https://github.com/steveloughran/winutils.git)

This will clone the repository. Find the folder **hadoop-3.0.0** and copy it to the C: drive. Make sure to rename this folder to **hadoop**.

A screenshot of a computer

Description automatically generated

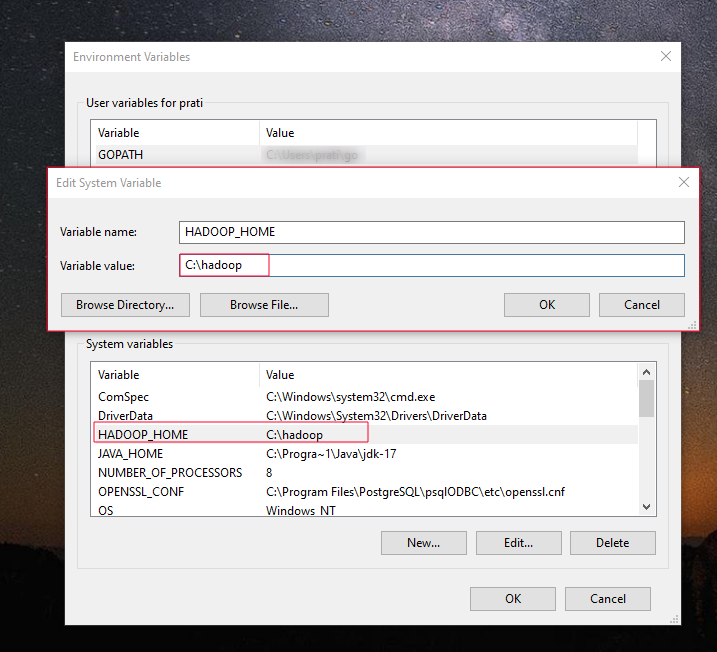
Next you need to set up the HADOOP\_HOME environment variable to tell Apache Spark where to find Hadoop.

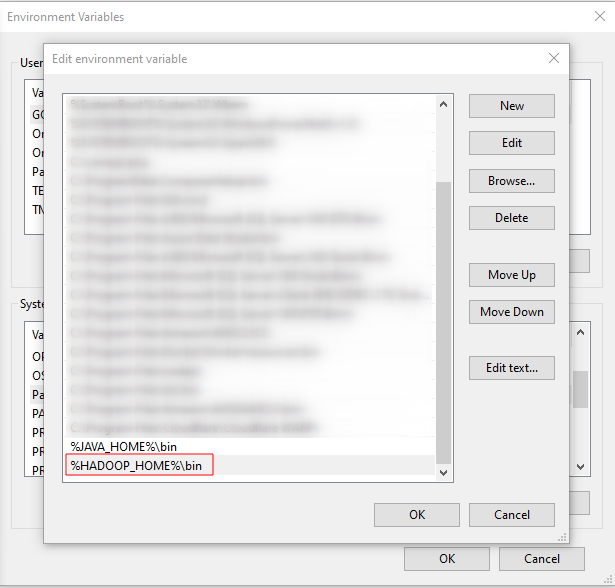
Start the "Advance Systems Settings" control panel in Windows:

A screenshot of a computer

Description automatically generated

Then create a new System Environment Variable called HADOOP\_HOME and set it to "C:\hadoop"





## 2. Creating the project in Eclipse

Start Eclipse, and create a new project. Then select Maven Project:

A screenshot of a computer

Description automatically generated

OR,

A screenshot of a computer program

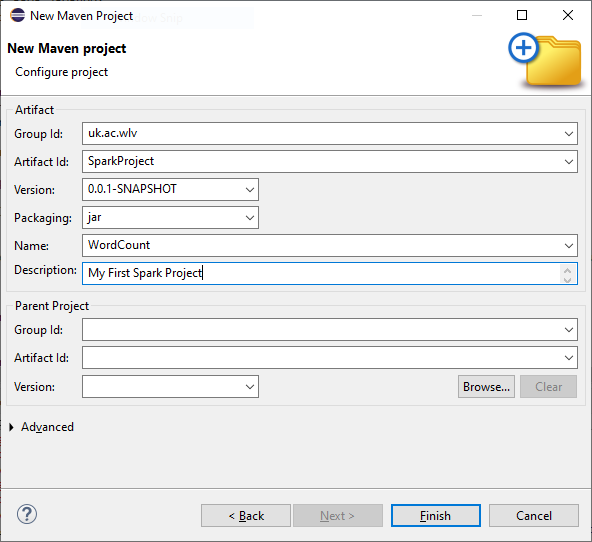
Description automatically generated

Then on the next page, make sure that the "simple project" is checked:

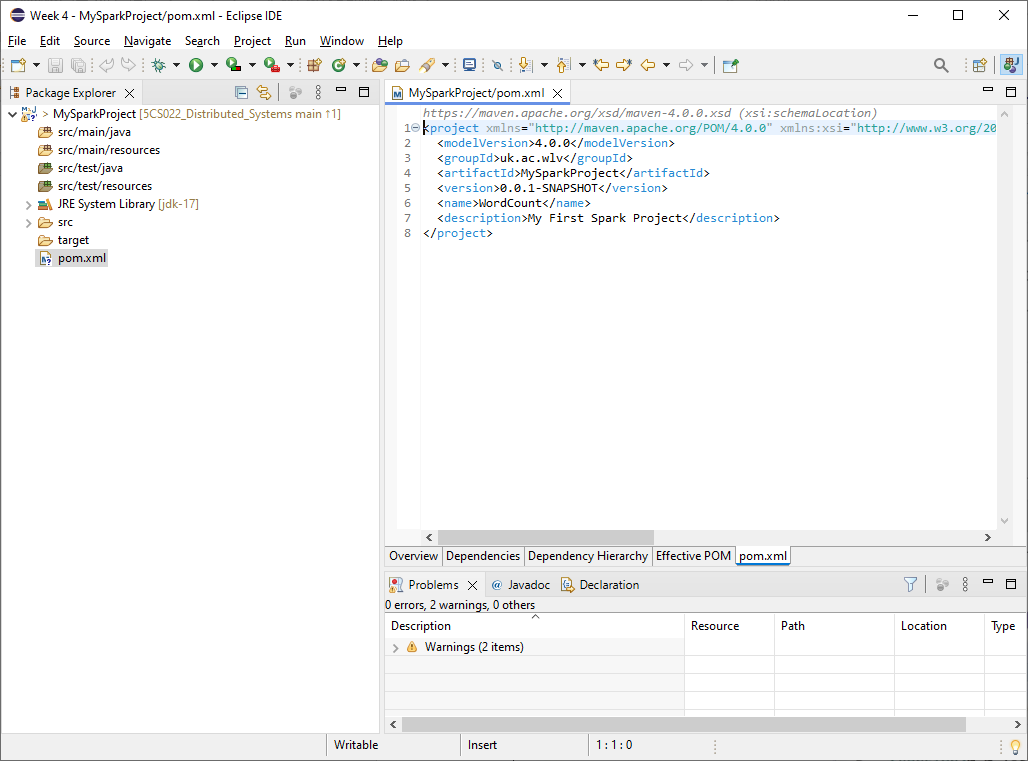
A screenshot of a computer

Description automatically generated

Then fill out the project information as follows:



When Eclipse has finished creating the project, open the "pom.xml" (This is the Maven project configuration file).



Replace it with the following:

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>uk.ac.wlv</groupId>

<artifactId>MySparkProject</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>WordCount</name>

<description>My First Spark Project</description>

<dependencies>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-core\_2.13</artifactId>

<version>3.5.1</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.13.0</version>

<configuration>

<source>${java.version}</source>

<target>${java.version}</target>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-jar-plugin</artifactId>

<version>3.3.0</version>

<configuration>

<archive>

<manifest>

<addClasspath>true</addClasspath>

<classpathPrefix>lib/</classpathPrefix>

<mainClass>uk.ac.wlv.WordCount</mainClass>

</manifest>

</archive>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-dependency-plugin</artifactId>

<executions>

<execution>

<id>copy</id>

<phase>install</phase>

<goals>

<goal>copy-dependencies</goal>

</goals>

<configuration>

<outputDirectory>${project.build.directory}/lib</outputDirectory>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

Save the file and then click the "run" button on the toolbar to update the configuration in Eclipse:

A screenshot of a computer

Description automatically generated

Then select the first "Maven build":

A screenshot of a computer

Description automatically generated

Then enter "eclipse:eclipse" for the "Goals" and click the Run button:

A screenshot of a computer

Description automatically generated

This will synchronise the Maven build configuration with Eclipse and bring Eclipse up to date.

## 3. Creating the WordCount program

Right click on the src/main folder in Eclipse and create a new Java class:

A screenshot of a computer program

Description automatically generated

Enter "uk.ac.wlv" for the package, and WordCount for the Name, and click finish.

Then replace the WordCount.java code with the following:

**package** uk.ac.wlv;

**import** java.io.IOException;

**import** java.util.Arrays;

**import** org.apache.hadoop.fs.FileSystem;

**import** org.apache.hadoop.fs.Path;

**import** org.apache.spark.SparkConf;

**import** org.apache.spark.api.java.JavaPairRDD;

**import** org.apache.spark.api.java.JavaRDD;

**import** org.apache.spark.api.java.JavaSparkContext;

**import** scala.Tuple2;

**public** **class** WordCount {

**public** **static** **void** main(String[] args) {

SparkConf sparkConf = **new** SparkConf();

sparkConf.setAppName("Spark WordCount example using Java");

/\* Tell Spark that we are running on this computer alone \*/

sparkConf.setMaster("local");

JavaSparkContext sparkContext = **new** JavaSparkContext(sparkConf);

/\* Reading input file \*/

JavaRDD < String > textFile = sparkContext.textFile("input.txt");

/\* This code snippet creates an RDD (Resilient Distributed Dataset) of words from each line of the input file and the flatMap function is used to split the text file into an ArrayList of words by applying the split(" ") method to each line, which separates the line into individual words. \*/

JavaRDD < String > words = textFile.flatMap(l -> Arrays.*asList*(l.split(" ")).iterator());

/\* Generate Pair of Word with count \*/

JavaPairRDD < String, Integer > pairs = words.mapToPair(w -> **new** Tuple2<String, Integer>(w, 1));

/\* Aggregate Pairs of Same Words with count \*/

JavaPairRDD < String, Integer > counts = pairs.reduceByKey((x, y) -> x + y);

/\* Deleting output directory if it already exists and saving the result file \*/

String outputPath = "output"; // Change this to your desired output directory

**try** {

FileSystem.*get*(sparkContext.hadoopConfiguration()).delete(**new** Path(outputPath), **true**);

} **catch** (IOException e) {

e.printStackTrace();

}

/\* Saving the result file \*/

**try** {

counts.saveAsTextFile(outputPath);

} **catch** (Exception e) {

e.printStackTrace();

}

/\* System.out.println(counts.collect()); \*/

System.***out***.println("Word Counts:");

**for** (Tuple2<String, Integer> tuple : counts.collect()) {

System.***out***.println(tuple.\_1() + ": " + tuple.\_2());

}

sparkContext.stop();

sparkContext.close();

}

}

## 4. Creating the input text file

Using Notepad (or similar), create a file called "input.txt" and fill it with some example English text. Save it in the same directory as the pom.xml file, that is, the project directory.

A screenshot of a computer program

Description automatically generated

## 5. Running the WordCount Spark program

Make sure that the file WordCount.java is opened as the current file in the Eclipse editor and click the run button on the Eclipse menu toolbar.

If it runs successfully, you should see the Spark logging output in Eclipse:



and there shouldn't be any error messages.

## 6. Debugging ERRORS

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

Exception in thread "main" java.lang.IllegalAccessError: class org.apache.spark.storage.StorageUtils$ (in unnamed module @0x6a03bcb1) cannot access class sun.nio.ch.DirectBuffer (in module java.base) because module java.base does not export sun.nio.ch to unnamed module @0x6a03bcb1

at org.apache.spark.storage.StorageUtils$.<clinit>(StorageUtils.scala:213)

at org.apache.spark.storage.BlockManagerMasterEndpoint.<init>(BlockManagerMasterEndpoint.scala:121)

at org.apache.spark.SparkEnv$.$anonfun$create$9(SparkEnv.scala:358)

at org.apache.spark.SparkEnv$.registerOrLookupEndpoint$1(SparkEnv.scala:295)

at org.apache.spark.SparkEnv$.create(SparkEnv.scala:344)

at org.apache.spark.SparkEnv$.createDriverEnv(SparkEnv.scala:196)

at org.apache.spark.SparkContext.createSparkEnv(SparkContext.scala:284)

at org.apache.spark.SparkContext.<init>(SparkContext.scala:483)

at org.apache.spark.api.java.JavaSparkContext.<init>(JavaSparkContext.scala:58)

at uk.ac.wlv.WordCount.main(WordCount.java:21)

If you ran into this issue. Right-click on the Java file >> “Run As” >> click on the run configurations.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

On the Arguments tab, add the following to the VM arguments:

**--add-opens java.base/sun.nio.ch=ALL-UNNAMED**

If you still run into an issue:

Right-click on **winutils.exe** file in the C:/hadoop/bin directory and “Run as Administrator”. You may come across this issue.

*The code execution cannot proceed because MSVCR100.dll was not found. Reinstalling the program may fix this problem.*

If error appears, follow the link below to download the Microsoft Visual C++ 2010 Service Pack 1 Redistributable Package. msvcr100.dll is a part of Microsoft Visual C++ and is required to run programs developed with Visual C++.

<https://www.microsoft.com/en-us/download/details.aspx?id=26999>

Even if you still receive the error message related to the Hadoop home not found or doesn’t exist, add the following to the VM arguments of your Java file.

--add-opens java.base/[sun.nio.ch](http://sun.nio.ch)=ALL-UNNAMED  
**-Dhadoop.home.dir=C:/hadoop  
-Djava.library.path=C:/hadoop/bin**

***The highlighted code is required to be added only if you have the error related to the Hadoop home path.***

## 6. View the output results

Assuming that your Spark program ran correctly, it would have created an output directory in your project directory:

A screenshot of a computer

Description automatically generated

Opening the "output" directory should give you the following files:

A screenshot of a computer

Description automatically generated

The results that we are looking for will be in the file "part-0000". Open that file in Notepad or any other text editors, and you should see a list of words and their counts.

A screenshot of a computer

Description automatically generated

## Tasks

1. Create a Spark program to count letters instead of words.