Programming Assignment 0

Creating Hadoop & Spark Clusters and Running a Simple Analytics Job

Due: February 1st, 2024 5:00PM
Submission: via Canvas, individual submission

Objectives
The goal of this programming assignment is to teach you how to:

- set up Hadoop & Spark
- perform HDFS operations
- run jobs on Spark cluster
- run a Word Count example using Spark

1. Introduction

This assignment is structured as a tutorial that will give you practical experience with Hadoop and Spark. First, you will learn how to setup a Hadoop and Spark configurations by running a standing-alone bash script. Next, you will be given instructions on how to start and stop HDFS service (used for your data) and Spark. After you successfully learn to run these services, you will be given commands for HDFS operations for storing, examining and modifying the data used in Spark in this and future assignments. Next, you will learn how to perform Spark jobs. After learning how to setup your Hadoop and Spark cluster and how to perform jobs, you will run a simple Word Count program using spark.

2. Setup Hadoop Cluster

In this section, you will first perform steps that will prepare your working environment for the main cluster setup. That includes learning how to remotely login to the department machines without requiring password every time and modifying environment variables, so your host knows where to find Hadoop and Spark, where to store intermediary results and logs, and where to look for the cluster configuration files.
In order to remotely connect to department machines please use ‘`ssh <username>@<hostname>`’, where `<username>` is your CSU-EID (e.g., `cjung`) and `<hostname>` is the machine you are connecting to (e.g., `denver.cs.colostate.edu`). Most students will already have ssh keys configured on the CS120 machines, if that’s not the case please contact the Tas.

Next, we want to add all the environment variables needed. Our department has implemented modules for different courses, making this easier for students. Few general commands for using modules include:

#load module named `<MODULE_NAME>` `module load <MODULE_NAME>`

#remove module named `<MODULE_NAME>` `module unload <MODULE_NAME>`

#remove all modules `module purge`

#show what commands a module `<MODULE_NAME>` implements `module show <MODULE_NAME>`

#list all loaded modules `module list`

For this assignment we will be using the module called '`courses/cs535/pa1`'. In order to automate this process so you don't need to load the module every time you use your cluster and for all machines, you will add directives in the `.bashrc` file to automatically load the module when you log in. We show how to do this in command line, but feel free to do this in GUI (copy/paste the two directives in the file).

# open your `.bashrc` file for editing, you can use other text editors too (Vim's commands are here) `vim ~/.bashrc`

# once in edit mode, add the following lines towards **the end of you file** (or copy/paste them in GUI)

```
source /etc/profile.d/modules.sh module
module purge
module load courses/cs535/pa1
```

#save the file and reload it in your shell (even if you edit the file in GUI)

```
source ~/.bashrc
```

NOTE: Please check if there are two or more `load <SOMETHING>` exist. If so, please comment out other `load` lines. Since `.bashrc` only allow one `load`, please comment out other load lines.

To check if this was successful, executing '`echo $HADOOP_HOME`' should output path to your Hadoop.

Note: if you use a different shell than bash, this process will be slightly different.

Note: if you do work on your local non-department machine, you will need to manually set the environment variables that modules implement. You can use the '`module show <MODULE_NAME>`' command to see them. For a line in the output '`setenv ENV_VAR_NAME /points/to/local/path`' the appropriate command you need to execute on your local machine will be '`export ENV_VAR_NAME=/points/to/local/path`'. To check if this was successful, executing '`echo $ENV_VAR_NAME`' should output '`/points/to/local/path`'

Next, we will unzip template Hadoop & Sparks configurations and run a bash script to automatically fill your machines and ports.

# unzip your template configurations.

```
unzip /s/$HOSTNAME/a/nobackup/cs535/<USERENAME_HERE>/<USERENAME_HERE>.zip -d ~/
```

# change directory to bigdata_configs

```
cd ~/bigdata_configs
```

# please read README.md before setting up the configurations.

```
cat README.md
```

# run to setup all configurations with a bash script. **Before running the script, please change the names of `hadoopConf` and `sparkConf` directory if you have and want to keep them.**

```
setup_all_config.sh
```

Note: After running the above command, you can see `hadoopConf` *and* `sparkConf` *that are automatically generated by the bash script.*

Note: These configuration files are customized for use on our department machines. In case you decide you want to run cluster on non-departmental machines, it is your responsibility to edit all the paths in the .xml files (for example for the temp directory and logs) as well as update the ports and hosts (some ports might be closed by external firewalls). Even if you have the resources for external cluster, this will add complexity, hence we recommend using the department machines for this course.

3. Running Hadoop Cluster

Now that your cluster is properly configured, it's time to learn how to start it, stop it, or check the operation status. To start HDFS, please login to your namenode (It is specified in your `~/bigdata_configs/README.md`). Make sure you are logged in to your name node and perform any of the following commands from this host.

The first time you use your cluster, you need to format HDFS. To do that, run:

```
hdfs namenode -format
```

Note: Only format your name node when setting up the cluster. Formatting it later will delete all your HDFS data!
Note for standalone mode: Since your machine was set the path for `hdfs`, use `$HADOOP_HOME/bin/hdfs namenode -format`

To start HDFS, run the start script:

```
start-dfs.sh
```

Once you start HDFS, in order to check the web portal, use your browser to open the URL `http://<namenode>:<port>` (as specified in the `dfs.namenode.http-address` property in `hdfs-site.xml`). Note if you are remotely connected to your namenode, you will need to create SSL port tunneling to be able to open the web interface from your local machine.

A simpler but less informative way is to check Java processes on your name node machine is using the command '`jps`' in terminal. You should see process '`NameNode`'. If you only see a process for '`jps`' when running this command from the name node host, then HDFS is not running.

To stop HDFS, run the stop script:

```
stop-dfs.sh
```

Once you are certain HDFS is properly setup, let's learn how to control your YARN resource manager process. First, log into your resource manager (the host specified in your `yarn-site.xml`). Make sure you are logged in to your resource manager and perform any of the following commands from this host. To start YARN, run the start script from your resource manager host:

```
start-yarn.sh
```

Once you start YARN, in order to check the web portal, use your browser to open the URL `http://<resource_manager_host>:<port>` (`<resource_manager_host>` is your resource manager as specified in your `yarn-site.xml`, and `<port>` is specified in your `yarn-site.xml`)
A simpler but less informative way is to check Java processes on your resource manager machine using the command '`jps`' in terminal. You should see process '`ResourceManager`'. If you only see a process for '`jps`' when running this command from the resource manager host, then YARN is not running.

To stop YARN, run the stop script:

```
stop-yarn.sh
```

Important note: make sure to stop all Hadoop services once you are done with using your cluster. When debugging, it's good idea to restart the services (order should be start HDFS, start YARN, stop YARN, stop HDFS).

Note for standalone mode: As the same reason of `hdfs`, use `$HADOOP_HOME/sbin/<script_name>.sh`

4. Work with HDFS

In order to run jobs on your Hadoop cluster, you need to make sure that the required data is in HDFS beforehand. We will go through couple commands here but please refer to the documentation for more exhaustive explanation on HDFS operations. Majority of these commands are based on the standard UNIX commands for working with file system, so it should be relatively straightforward.

To see the content of your root directory in HDFS run

```
hadoop fs -ls /
```

Note: the '`/`' has to be included in the command. If you have a directory named '`example`' in your root HDFS directory, the HDFS path will be '`/example`'. What happens if you run '`hadoop fs -ls`'?

To create a folder named '`example`' in your root HDFS directory run

```
hadoop fs -mkdir /example
```

To see content in folder '`example`' in your root HDFS directory run

```
hadoop fs -ls /example
```

To upload data to HDFS run

```
hadoop fs -put <SOURCE> <DESTINATION>
```

where `<SOURCE>` is the data path on your machine and `<DESTINATION>` is the location in HDFS. Make sure you don't forget to include the leading '`/`' when specifying the destination path.

Note for standalone mode: As the same reason of `hdfs`, use `$HADOOP_HOME/bin/hadoop <rest_of_commands>` for all `hadoop` command.

5. Running Spark Cluster

- To be safe, start Hadoop cluster before launching Spark cluster (no need to start YARN)
- To start the cluster, login to Spark master node which is the same node as Hadoop's namenode:

```
~$ start-master.sh
~$ start-workers.sh
```

- Check master webUI using <SPARK_MASTER_IP>:<SPARK_MASTER_WEBUI_PORT>

  To check if a spark session can be created, run the following command

```
~$spark-shell
```

  If there are no errors, you can create a spark cluster!

- To stop the cluster:
  - Login to Spark master node.
  - Run following scripts to launch Master as well as workers

```
~$ stop-master.sh
~$ stop-workers.sh
```

6. Launching Spark applications

- Spark applications can be launched using spark-submit script.

- Change directory to your project folder.
- Run following command with appropriate values.

```
~$  spark -submit  --class  <your  Class>  --master  <your  Master>  <yourJar>
<any_arguments (in the Hadoop file system)>
```

You can refer http://spark.apache.org/docs/latest/submitting-applications.html for more information.

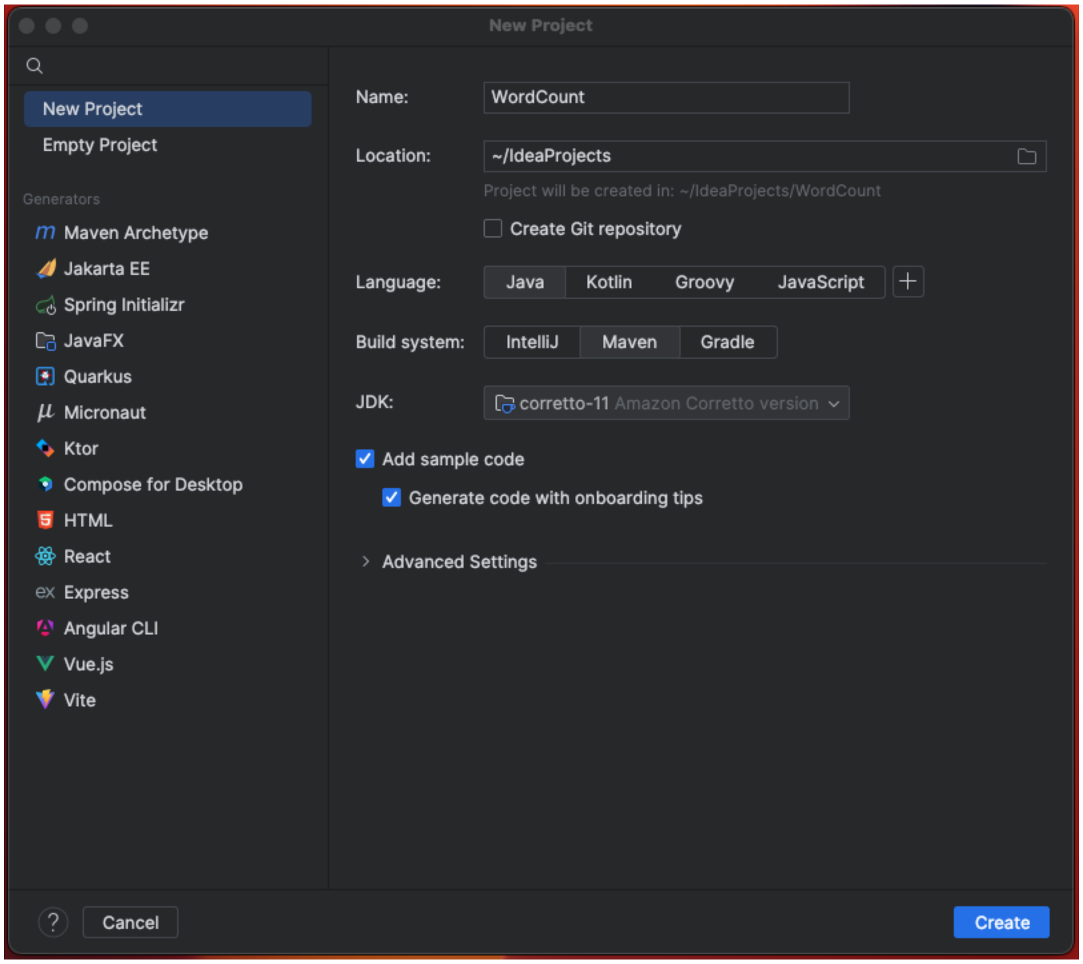
7. Run an example on Spark Cluster

Now, Hadoop and Spark are ready, and we will use the word-count example to run spark cluster. To make a .jar file and run it in your Spark cluster, please follow these steps:

1. Install IntelliJ: https://www.jetbrains.com/idea/download/
2. Signup and link with your school email to get 1 year education license.
3. Open it IntelliJ.
4. Create a project which name is "WordCount" and put any path where you want to save the project in "Location". Set "Language" to "Java" and select "Maven" for "Build System". In "JDK" section, click "Download JDK", and set "Version" as "11" and Vendor to "Amazon Corretto". (If you already have it, you don't have to re-download) Please check the **figure 1** below.
5. After creating the project, you can find "pom.xml". Please download our "pom.xml" and update the original one to ours. Then, click the update button to install packages if it is needed. (You can find the button in the **figure 2**.
6. Then you can see a Spark installation button at right-bottom corner. Please click that button to see the installation UI as **figure 3** and press the "install" button.
7. When the installation is done, replace the contents of "main.java" to our "WordCount.java" and rename it as "WordCount.java".
8. If you want to skip the running spark job in your machine, please move to step # 14.
9. Download our input.txt and put the file in 'java' directory on the left panel by dragging the input.txt.
10. To run the example in your local machine, it needs a path for input. Please click the "current file" button at the top-right corner and hit the "Edit Configurations" button as **figure 4**.
11. Click the "+" button and choose "application".
12. Set the name and class as **figure 5** and put the corresponding input path. Press "OK"
13. Click the "run" button at the top-right corner to run the program. At the bottom, you can see the progress and results if you scroll it up.
14. From now on, we will generate .jar file and move it to your account's directory.
15. Please remove "`.setMaster("local")`" in WordCount.java file.
16. Click the "Maven" button in the middle-right corner. You can see "package" button if you click "Lifecycle" directory.
17. Double-click the "package" button to execute it. It will generate WordCount-1.0-SNAPSHOT.jar in "WordCount/src/target" directory. (Please check with the **figure 6** below)
18. From now on, we will run the .jar file using lab machines.
19. Access to your master node (=namenode).
20. Create "cs535" directory in your home directory as `mkdir ~/cs535`

21. Create “PA0” directory in “cs535” directory as `mkdir ~/cs535/PA0`
22. Please copy your “WordCount” directory from your machine to “~/cs535/PA0” in your account.
23. Change directory to “~/cs535/PA0/WordCount”
24. Download “input.txt” from Canvas and put it in the current path.
25. Make sure that your dfs cluster and spark cluster are running with `jps` command. If you can’t see “Namenode” and “Master”, please run `start-dfs.sh`, `start-master.sh`, and `start-workers.sh` in order.
26. Put your “input.txt” to hdfs as `hadoop fs -put ./input.txt /`
27. Please run the spark job with this command:
    ```
    spark-submit --class
    org.example.WordCount --master spark://<master node>:<master port>
    ~/cs535/PA0/WordCount/target/WordCount-1.0-SNAPSHOT.jar /input.txt
    ```
28. If the example finished successfully, you can see the results in the terminal. Please save the results as “output.txt” and put it in “~/cs535/PA0/WordCount”.
29. Change directory to ““~/cs535/PA0/” as `cd ..` or `cd "~/cs535/PA0/`.
30. Please tar “WordCount” using this command: `tar -cvf <USERNAME>.tar WordCount`
31. Submit the tar file on Canvas.

Note: you can find the master node and port in “~/sparkConf/spark-env.sh”.



Name: WordCount

Location: ~/IdeaProjects

Project will be created in: ~/IdeaProjects/WordCount

☐ Create Git repository

Language: [✓ Java] Kotlin Groovy JavaScript

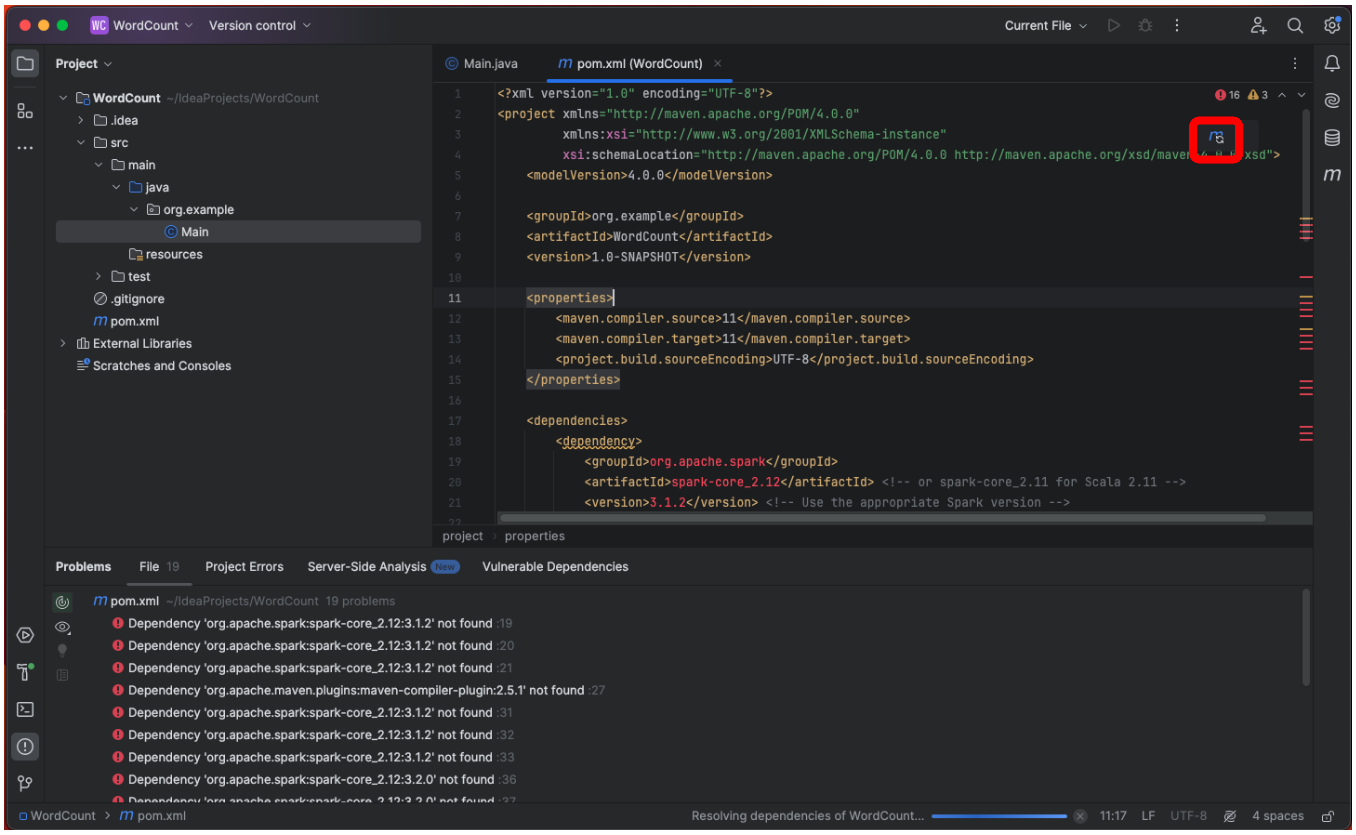
Build system: IntelliJ [✓ Maven] Gradle

JDK: corretto-11 Amazon Corretto version

☑ Add sample code

☑ Generate code with onboarding tips

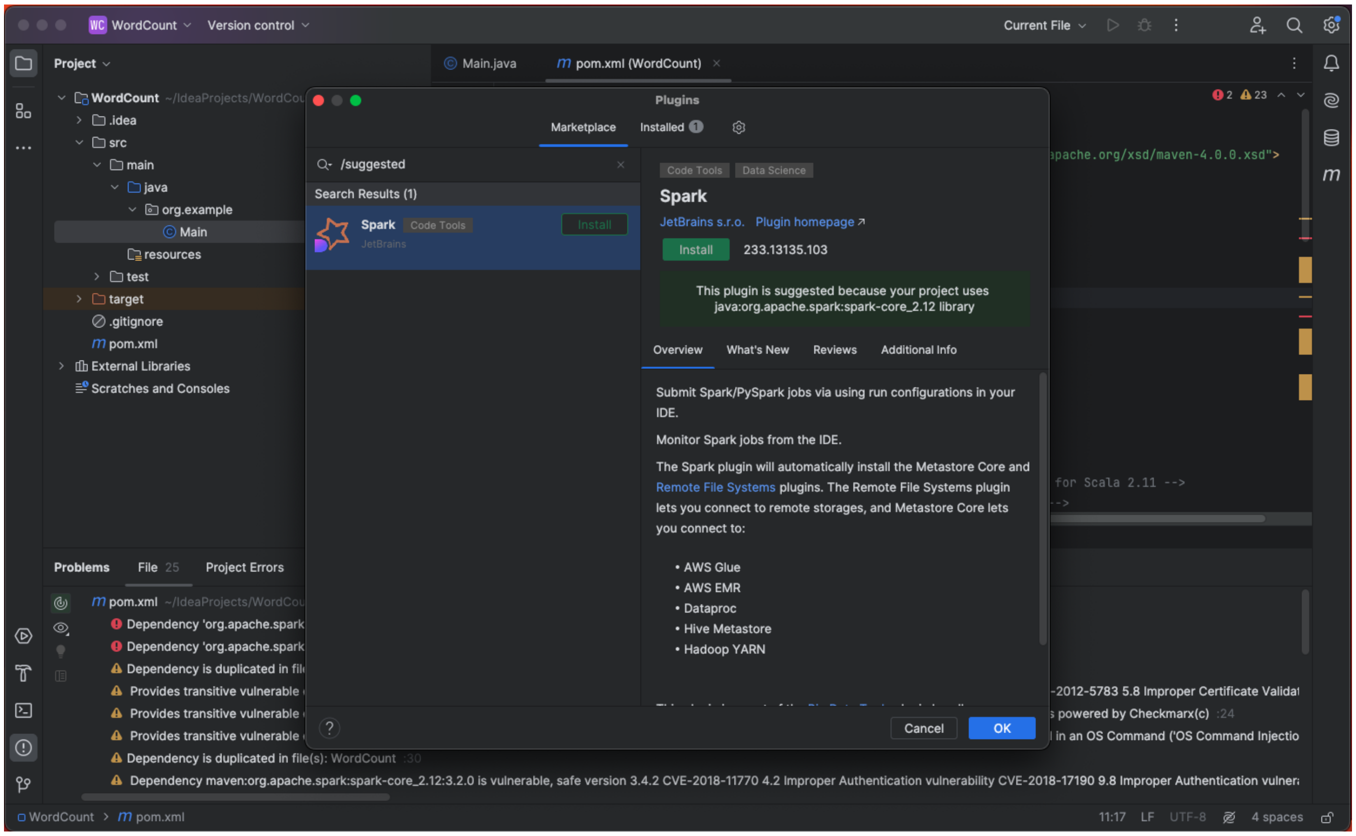
Figure 1. Initial setting for creating a project.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<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 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>WordCount</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-core_2.12</artifactId> <!-- or spark-core_2.11 for Scala 2.11 -->
            <version>3.1.2</version> <!-- Use the appropriate Spark version -->
```

```
pom.xml ~/IdeaProjects/WordCount 19 problems
  Dependency 'org.apache.spark:spark-core_2.12:3.1.2' not found :19
  Dependency 'org.apache.spark:spark-core_2.12:3.1.2' not found :20
  Dependency 'org.apache.spark:spark-core_2.12:3.1.2' not found :21
  Dependency 'org.apache.maven.plugins:maven-compiler-plugin:2.5.1' not found :27
  Dependency 'org.apache.spark:spark-core_2.12:3.1.2' not found :31
  Dependency 'org.apache.spark:spark-core_2.12:3.1.2' not found :32
  Dependency 'org.apache.spark:spark-core_2.12:3.1.2' not found :33
  Dependency 'org.apache.spark:spark-core_2.12:3.2.0' not found :36
```

Figure 2. Updating button for Maven



This plugin is suggested because your project uses java:org.apache.spark:spark-core_2.12 library

Submit Spark/PySpark jobs via using run configurations in your IDE.

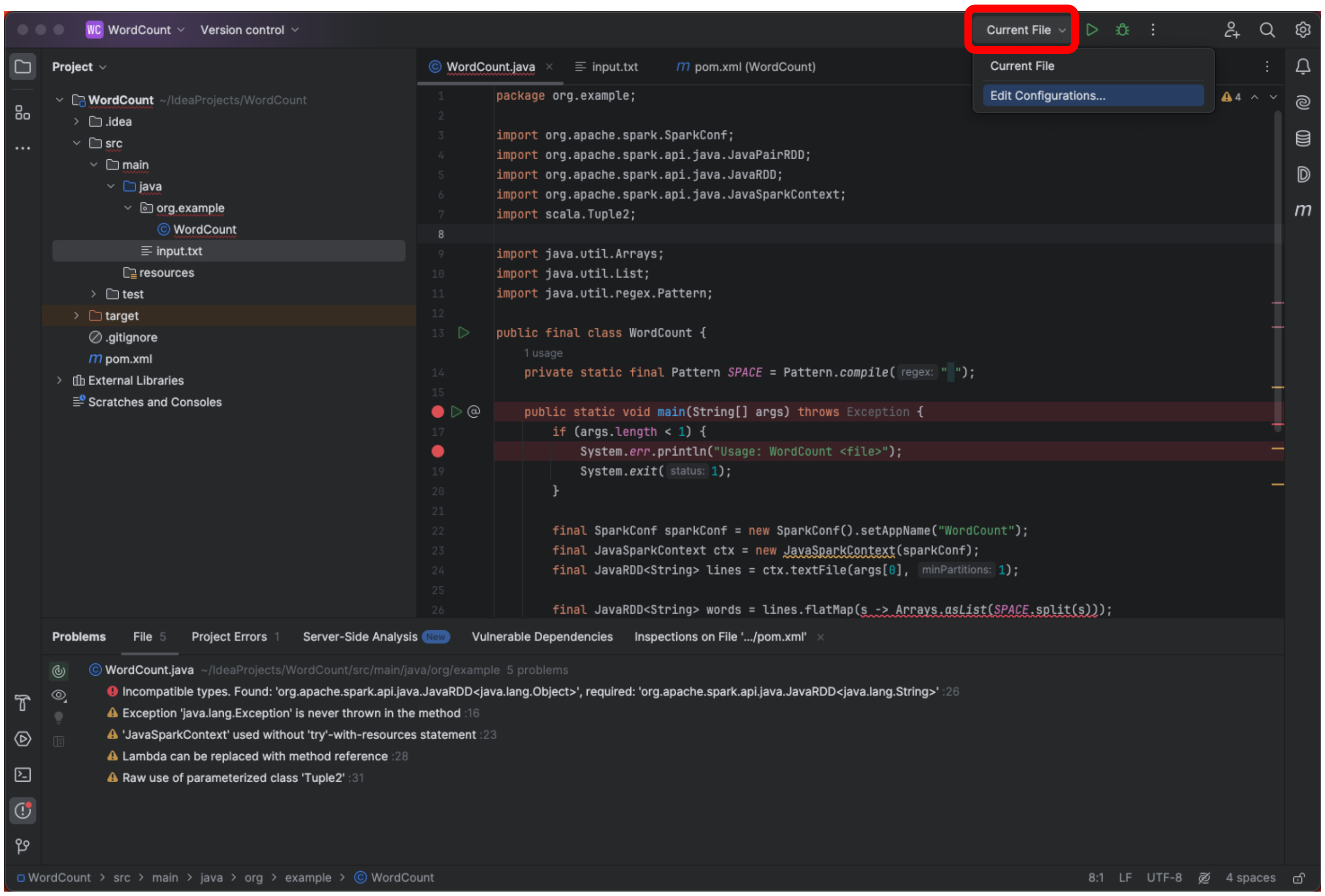
Monitor Spark jobs from the IDE.

The Spark plugin will automatically install the Metastore Core and Remote File Systems plugins. The Remote File Systems plugin lets you connect to remote storages, and Metastore Core lets you connect to:

- AWS Glue
- AWS EMR
- Dataproc
- Hive Metastore
- Hadoop YARN

```
pom.xml ~/IdeaProjects/WordCou
Dependency 'org.apache.spark
Dependency 'org.apache.spark
Dependency is duplicated in fil
Provides transitive vulnerable
Provides transitive vulnerable
Provides transitive vulnerable
Dependency is duplicated in file(s): WordCount :30
Dependency maven:org.apache.spark:spark-core_2.12:3.2.0 is vulnerable, safe version 3.4.2 CVE-2018-11770 4.2 Improper Authentication vulnerability CVE-2018-17190 9.8 Improper Authentication vulnerai
```

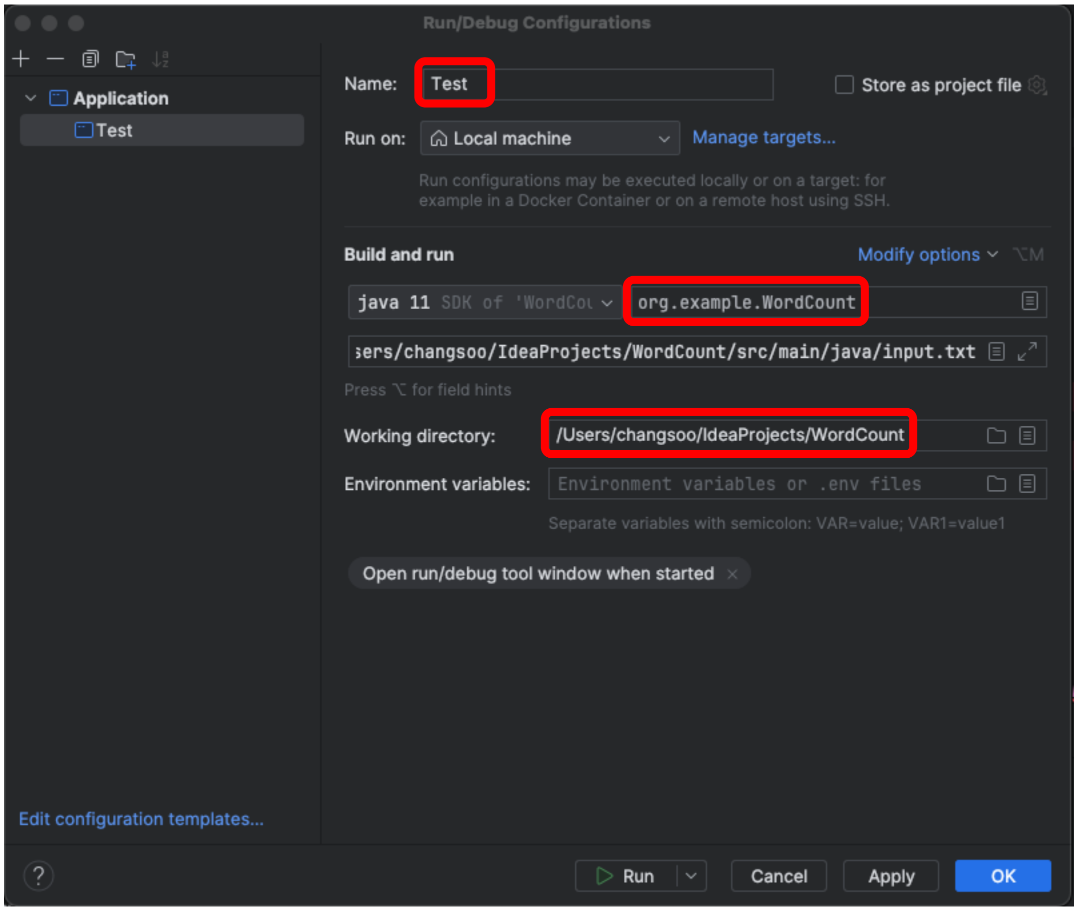
Figure 3. Spark Installation UI



```java
package org.example;

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;

import java.util.Arrays;
import java.util.List;
import java.util.regex.Pattern;

public final class WordCount {
    private static final Pattern SPACE = Pattern.compile(" ");

    public static void main(String[] args) throws Exception {
        if (args.length < 1) {
            System.err.println("Usage: WordCount <file>");
            System.exit(1);
        }

        final SparkConf sparkConf = new SparkConf().setAppName("WordCount");
        final JavaSparkContext ctx = new JavaSparkContext(sparkConf);
        final JavaRDD<String> lines = ctx.textFile(args[0], 1);

        final JavaRDD<String> words = lines.flatMap(s -> Arrays.asList(SPACE.split(s)));
```

```
WordCount.java ~/IdeaProjects/WordCount/src/main/java/org/example 5 problems
  Incompatible types. Found: 'org.apache.spark.api.java.JavaRDD<java.lang.Object>', required: 'org.apache.spark.api.java.JavaRDD<java.lang.String>' :26
  Exception 'java.lang.Exception' is never thrown in the method :16
  'JavaSparkContext' used without 'try'-with-resources statement :23
  Lambda can be replaced with method reference :28
  Raw use of parameterized class 'Tuple2' :31
```

Figure 4. Editing configurations for testing on local



Name: Test

Run on: Local machine

Run configurations may be executed locally or on a target: for example in a Docker Container or on a remote host using SSH.

Build and run

java 11 SDK of 'WordCou

org.example.WordCount

sers/changsoo/IdeaProjects/WordCount/src/main/java/input.txt

Press ⌥ for field hints

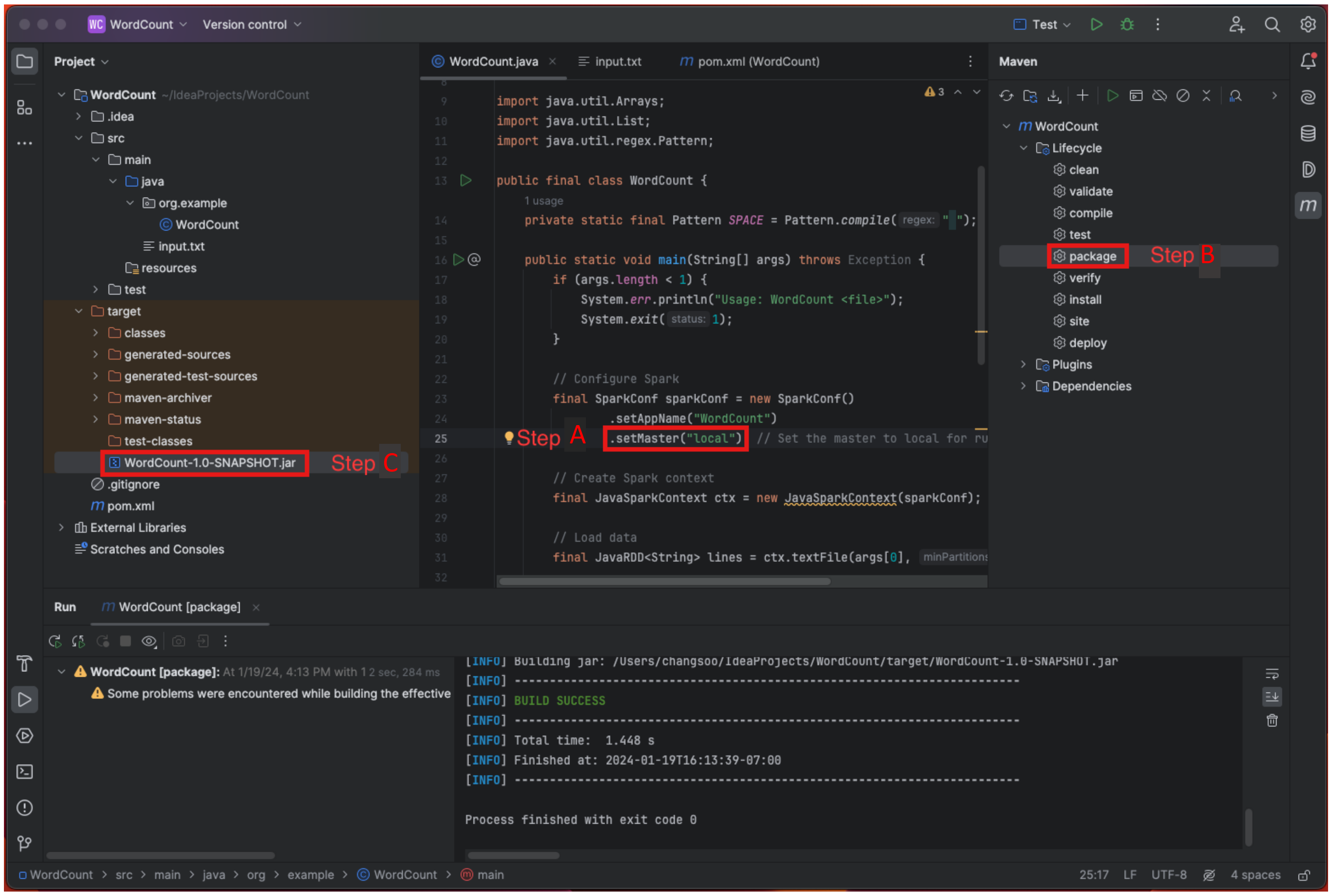
Working directory: /Users/changsoo/IdeaProjects/WordCount

Environment variables: Environment variables or .env files

Separate variables with semicolon: VAR=value; VAR1=value1

Open run/debug tool window when started

Figure 5. Details of the testing configurations



```
import java.util.Arrays;
import java.util.List;
import java.util.regex.Pattern;

public final class WordCount {
    1 usage
    private static final Pattern SPACE = Pattern.compile( regex: " ");

    public static void main(String[] args) throws Exception {
        if (args.length < 1) {
            System.err.println("Usage: WordCount <file>");
            System.exit( status: 1);
        }

        // Configure Spark
        final SparkConf sparkConf = new SparkConf()
                .setAppName("WordCount")
                .setMaster("local") // Set the master to local for ru

        // Create Spark context
        final JavaSparkContext ctx = new JavaSparkContext(sparkConf);

        // Load data
        final JavaRDD<String> lines = ctx.textFile(args[0], minPartitions
```

```
WordCount [package]: At 1/19/24, 4:13 PM with 1 2 sec, 284 ms
  Some problems were encountered while building the effective
```

```
[INFO] Building jar: /Users/changsoo/IdeaProjects/WordCount/target/WordCount-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.448 s
[INFO] Finished at: 2024-01-19T16:13:39-07:00
[INFO] ------------------------------------------------------------------------

Process finished with exit code 0
```

Figure 6. Generating .jar file

8. Submission

This assignment requires an **individual submission**. Please submit your tarball of "WordCount" directory via Canvas. Do not miss any file. You will need to rerun the example during demo. For your demo, you are not allowed to use any file outside of your Canvas submission except the Hadoop and Spark software.

9. Grading

Each of the submissions will be graded based on the demonstration of your software to GTA. During the demonstration, you should present your machines and their roles and run the example on your Spark cluster. (2 points)
Demo includes short interviews about your software design and implementation details. Each question and answer will count in your score. (3 points) This assignment will account for 5% of your final grade. The grading will be done on a 5-point scale.

You are required to work alone for this assignment.

Late policy

Please check the late policy posted on the Canvas course page.