# Lab 5

The objective of this laboratory is to start playing around with Apache Spark. We provide to you a template[1] in Java, from which you can easily write your first application. As a first task, you can repeat the exercise you had in Lab 2, which is repeated below for your convenience. As we have always suggested, you can also experiment on your own variations on the theme.

The workflow for building is the same as the one we used with Hadoop: once unzipped the template, open the project in your IDE (Eclipse), modify it, and then export a jar. Then, submit this job either locally[2][3]:

```
spark-submit  --class it.polito.bigdata.spark.example.SparkDriver
--deploy-mode client --master local SparkProject.jar arguments
```

or on the cluster:

```
spark-submit  --class it.polito.bigdata.spark.example.SparkDriver
--deploy-mode cluster --master yarn SparkProject.jar arguments
```

Notice how in the template we did not, even if possible, configure the Spark master and other settings explicitely in the main class of the driver. In this way, you can always submit the same jar on any configuration, either Spark standalone locally or a Yarn cluster. The main advantage of this method is for testing your application: first you can try your application on your machine, on a sample file, and then safely submit the same application for the real work.

## 1. Filter a file

If you completed Lab 1, you should now have (at least one) huge files with the word frequencies in the amazon food reviews, in the format `word\tnumber`, where number is an int (a copy of the output of Lab 1 is available in the HDFS shared folder /data/students/bigdata-01QYD/Lab2/). You should also have realized that inspecting these results manually is not feasible.

---

[1] http://dbdmg.polito.it/wordpress/wp-content/uploads/2016/05/Lab5_Template.zip

[2] You can specify the number of CPUs like this: --master local[2] or --master local[*] to use all the CPUs of your machine. To test concurrency, you should at least have 2 CPUs.

[3] To read from a local file, specify the full path like: “file:///absolute/path/of/the/file”

Your task is to write a **Spark** application to filter these results, and analize the filtered data. The filter we propose to you is the following: try to keep only the words that start with “ho”. How large is the result of this filter? Do you need to filter more?

Can you specify the beginning string (“ho”) as a command-line parameter?

# Bonus questions

Explore the web dashboard of your jobs[4]. Check out the DAG of your RDDs and the statistics for the application, then answer the following questions:

- How many executors have been allocated to your job?
- How was the input data splitted among them?
- How much memory each of these partitions took? And what was the total memory available on their executors?
- How much data was shuffled? Why?
- Was your application well balanced among the executors, or were there tasks much slower/heavier/bigger than the others? Why?

Keep in mind these questions for the next labs as well.

[4] For local applications, the web dashboard is killed as soon as your job finishes. On the cluster, it is moved on a history server, so you can access it also after the job has completed. The history is cleared after a fixed time (in our case, one week).