# SPARK Cluster – How to setup and use

## Setting up OS Environment

### On Windows

* HADOOP\_HOME >> C:\hadoop  
  **note**: On windows we need to set up HADOOP, because windows needs HADOOP executer, which is not available in SPARK binary files, so you can download the same version used in SPARK, then add the [executer file](https://github.com/srccodes/hadoop-common-2.2.0-bin/archive/master.zip).
* JAVA\_HOME >> C:\Program Files\Java\JAVA\_VERSION
* SPARK\_HOME >> C:\spark

### On Uinux

* JAVA\_HOME >> C:\Program Files\Java\JAVA\_VERSION
* SPARK\_HOME >> C:\spar

## Setting up IDE settings "Eclipse"

* Create a new JAVA project.
* Convert project to Maven project.
* Go to POM, then add new Dependencies from [download page](http://spark.apache.org/downloads.html).

## Java Code

conf.setAppName("Count Word V1.34"); //Setting the application name. conf.setMaster("spark://192.168.1.136:7077"); //Specify the master machine

//The next line will specify the number of cores that will be used by the application in total. So if we used 4 and we have 4 Workers, then each worker will use 1 core.

conf.set("spark.cores.max", "4");

//Then we pass the configuration to the SparkContext in order to start the Driver Program.

JavaSparkContext context = **new** JavaSparkContext(conf);

ArrayList<String> data = **new** ArrayList<>();

data.add("hasan is hasan, he is a hasan.");

data.add("hasan is hasan, he is a Khamees.");

data.add("hasan is hasan, he is a hasan.");

//Using Partitions means that you will devide your data into "x" chunks, why? Because let's say that you have a Huge data with moderate workers, then the number of default partitioned data could cause memory fault. So we can use this formula to calculate partitions (NumberOfWorkers\*NumberOfCores) in order to get the best of the cluster.

JavaRDD<String> lines = context.parallelize(data).repartition(2);

//But when we use coalesce, then we pass the number of partitions with the ability of shuffling the RDD chunks among partitions. For example, if we have 100 node, but when we partitioned the data to 4 partitions, then 94 worker won't work, but when we used coalesce then the chunks will be shuffled on all the 100 workers.

JavaRDD<String> lines = context.parallelize(data).coalesce(4,true);

//This code is required, because the Driver needs the executed files in order to pass executors to the Workers, because the Workers don't have permission on the files. And any file is used in the program will be added using context.addFile(path);

context.addJar("target/NewSP-0.0.1-SNAPSHOT.jar");

## Setting Up and Starting the Cluster

There is 3 ways you can deploy Cluster with SPARK:

* Standalone Cluster, built in SPARK – I will use this cluster.
* APACHE MESOS "Kernel that can provide APIs to manage resources that would be used by spark"
* HADOOP YARN "Hadoop architectural center that allow resource management"

### On Windows

On Windows we need to start MASTER machine and WORKER machine by hand by writing the next commands in SPARK/bin directory:

#ON THE MASTER MACHINE, WRITE THIS COMMAND

spark-class org.apache.spark.deploy.master.Master

#ON EACH WORKER, RUN THIS COMMAND TO CONNECT THE WORKER WITH THE MASTER

spark-class org.apache.spark.deploy.worker.Worker spark://IP:7077

#OR YOU CAN USE THIS COMMAND TO RUN WORKER IN ORDER TO SPECIFY THE NUMBER OF CORES WILL BE ASSIGNED TO WORK WITH MASTER

spark-class org.apache.spark.deploy.worker.Worker spark://IP:7077 –c 2

### On Unix

* Go to conf/spark-env.sh, then set the SPARK configuration using these keywords:

SPARK\_MASTER\_HOST >> \*Set the master IP\*

SPARK\_MASTER\_PORT >> \*Set the master port, usually 7077\*

SPARK\_WORKER\_CORES >> \*Set the default workers' core\*

* Then, Go to conf/slaves file, and add the workers IPs:

#for Example:

192.168.1.3

192.168.1.4

192.168.1.5

* Finally, Use the Spark S-SHELL in order to run the master and workers:

#this command will start the master.

Sbin/start-master.sh

#this command will start the slaves instances from Slave file

Sbin/start-slaves.sh

#starts both

Sbin/start-all.sh

## Run your code "Processing Steps"

* If you run the code now, the Spark Driver will star with the specified configurations.
* The driver will split data and distribute it on Workers "Slaves" as Executors.
* Workers use the data from Executors and then will return data to the Driver.
* Driver will reduce the data.