Apache Spark Tutorial–Run your First Spark Program

REQUEST INFO

What will you learn from this Spark Tutorial?

In this spark scala tutorial you will learn-

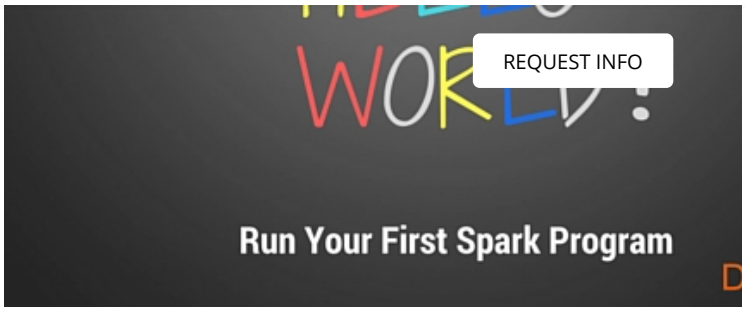
- Steps to install spark
- Deploy your own Spark cluster in standalone mode.
- Running your first spark program : Spark word count application.

Pre-requisites to Getting Started with this Apache Spark Tutorial

Before you get a hands-on experience on how to run your first spark program, you should have-

i. Understanding of the entire Apache Spark Ecosystem

ii. Read the Introduction to Apache Spark tutorial



Modes of Apache Spark Deployment

Before we begin with the Spark tutorial, let's understand how we can deploy spark to our systems –

1. **Standalone Mode in Apache Spark**

Spark is deployed on the top of Hadoop Distributed File System (HDFS). For computations, Spark and MapReduce run in parallel for the Spark jobs submitted to the cluster.

2. **Hadoop YARN/ Mesos**

Apache Spark runs on Mesos or YARN (Yet another Resource Navigator, one of the key features in the second-generation Hadoop) without any root-access or pre-installation. It integrates Spark on top Hadoop stack that is already present on the system.

3. **SIMR (Spark in Map Reduce)**

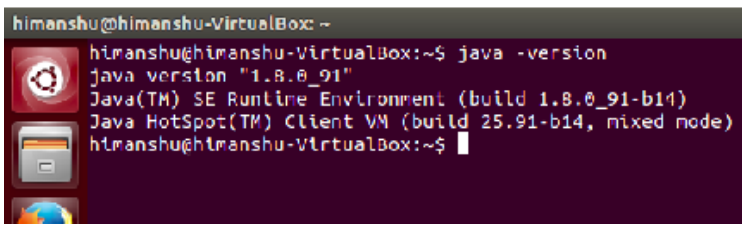
This is an add-on to the standalone deployment where Spark jobs can be launched by the user and they can use the spark shell without any

Getting Started with Apache Spark Standalone Mode of Deployment

Step 1: Verify if Java is installed

Java is a pre-requisite software for running Spark Applications. Use the following command to verify if Java is installed -

```
$java -version
```



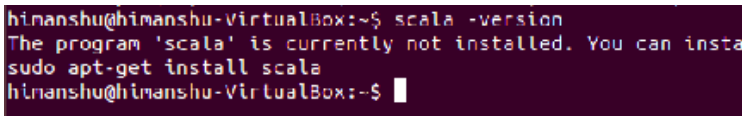
The above screenshot shows the version details of the Java installed on the machine. In case Java is not installed then head on to our Hadoop Tutorial for Installation. Follows the steps listed under "Install Java" section of the Hadoop Tutorial to proceed with the Installation.

Step 2 – Verify if Spark is installed

As Apache Spark is used through Scala programming language, Scala should be installed to proceed with installing spark cluster

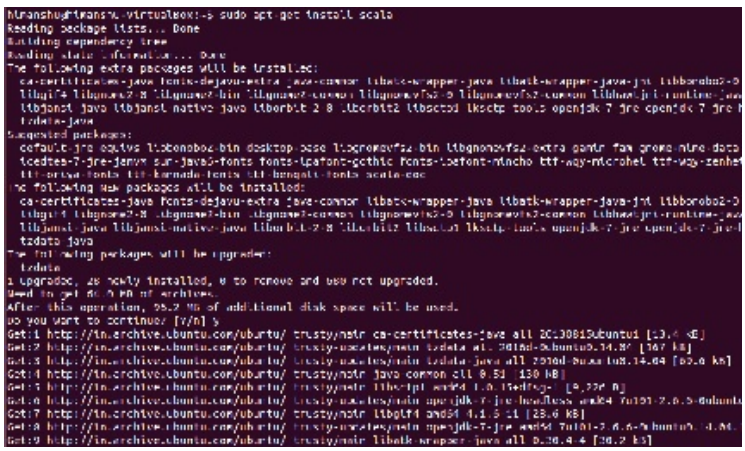
jersey

- Hadoop Training in Dallas
- Hadoop Training in Atlanta
- Hadoop Training in Chicago
- Hadoop Training in Canada
- Hadoop Training in Charlotte
- Hadoop Training in Abudhabi
- Hadoop Training in Dubai
- Hadoop Training in Detroit
- Hadoop Training in Edison
- Hadoop Training in Germany
- Hadoop Training in

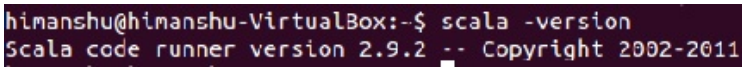


In case Scala is already installed on your system, it will display the version details. In the above screenshot, Scala programming language is not installed on my system. Let's install Scala first -

```
$sudo apt-get install scala
```



Verifying the installation



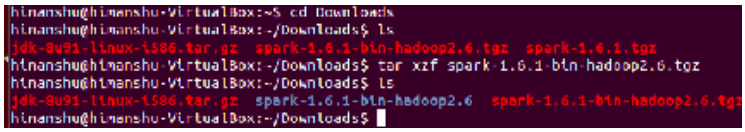
Step 3: Download and Install Apache Spark:

Download the latest version of Apache Spark

downloads folder. To install Spark, extract the tar file using the following comma

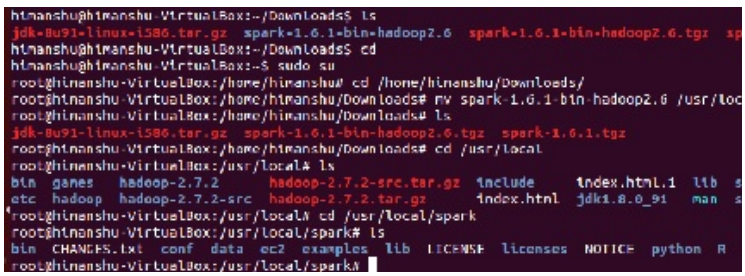
(In this spark tutorial, we are using spark-1.3.1-bin-hadoop2.6 version)

```
$ tar xvf spark-1.6.1-bin-
hadoop2.6.tgz
```



Move the spark downloaded files from the downloads folder to your local system where you plan to run your spark applications. Use the commands:

```
$ sudo su –
Password:
# cd /home/himanshu/Downloads/
# mv spark-1.6.1-bin-hadoop2.6
/usr/local/spark
# exit
```



Spark Configuration

Let's setup the environment variable for Apache

```
/bin
```

REQUEST INFO

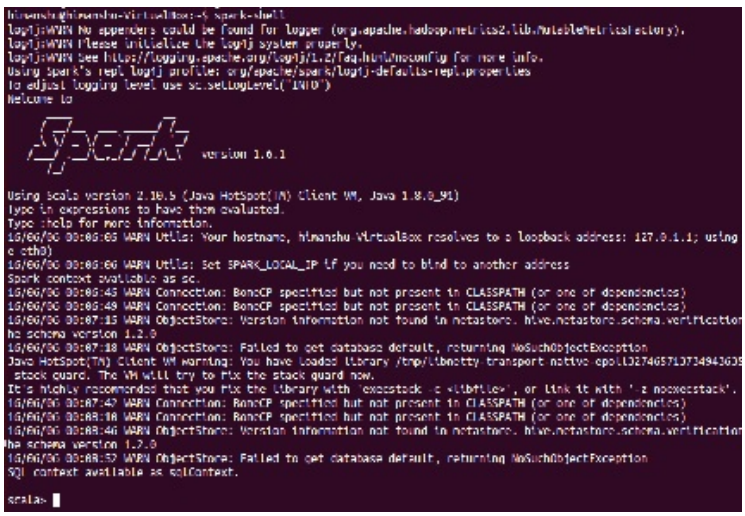
We add the above line *~/.bashrc* file and save it. Setting the PATH variable will locate the Spark executables in the location /usr/local/spark/bin.

Step 3: Verify Apache Spark installation

Verify the installation using the following command:

```
$spark-shell
```

In case the installation happened successfully, the above command will start Apache Spark in Scala.



In other words, for this, we just have to place the compiled version of Apache Spark

The most common way to REQUEST INFO applications on the cluster is to use the shell command spark-submit. When using spark-submit shell command the spark application need not be configured particularly for each cluster as the spark-submit shell script uses the cluster managers through a single interface. Spark-submit script has several flags that help control the resources used by your Apache Spark application. Spark-submit flags dynamically supply configurations to the Spark Context object.

Let's look at the same hadoop MapReduce example of Word Count in Apache Spark as well-

The input in the file input.txt contains the following text -

In as name to here them deny wise this. As rapid woody my he me which. Men but they fail shew just wish next put. Led all visitor musical calling nor her. Within coming figure sex things are. Pretended concluded did repulsive education smallness yet yet described. Had countryman his pressed shewing. No gate dare rose he. Eyes year if miss he as upon.

Below is the source code for the Word Count program in Apache Spark -

```
import org.apache.spark.SparkContext
```

```
   def main(args: Array[String]) {
                                    REQUEST INFO

    val sc = new SparkContext(
"local", "Word Count", "/usr/local
/spark", Nil, Map(),
Map())
      val input =
sc.textFile("input.txt")
      Val count = input.flatMap(line
⇒ line.split(" "))
      .map(word ⇒ (word, 1))
      .reduceByKey(_ + _)
      count.saveAsTextFile("outfile")
      System.out.println("OK");
   }
}
```

Let's understand the word count example in Spark step by step –

Linking with Apache Spark

The first step is to explicitly import the required spark classes into your Spark program which is done by adding the following lines -

import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark._

Creating a Spark Context Object

The next step is to create a Spark context object with the desired spark configuration that tells Apache Spark on how to access a cluster. The below line of code in the word count example

"Word Count" – This is the application that you want to run.

"local"- This parameter denotes the master URL to connect the spark application to.

/usr/local/spark- This parameter denotes the home directory of Apache Spark.

Map() – The first map specifies the environment whilst the second one specifies the variables to work nodes.\

Creating a Spark RDD

The next step in the Spark Word count example creates an input Spark RDD that reads the text file input.txt using the Spark Context created in the previous step-

val input = sc.textFile("input.txt")

Spark RDD Transformations in Wordcount Example

The below lines of spark application code transform the input RDD to count RDD -

```
Val count = input.flatMap (line ⇒ line. Split (" "))
    .map (word ⇒ (word, 1))
    .reduceByKey (_ + _)
```

In the above piece of code, flatMap () is used to tokenize the lines from input text file into words.

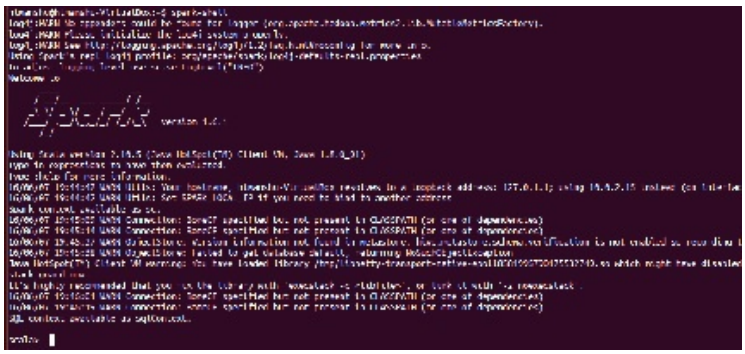
Map () method counts the frequency of each word.

Application

We will submit the word count example in Apache Spark using the Spark shell instead of running the word count program as a whole -

Let's start Spark shell

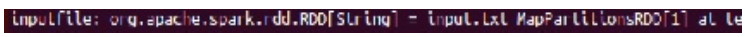
```
$ Spark-shell
```



Let's create a Spark RDD using the input file that we want to run our first Spark program on. You should specify the absolute path of the input file-

```
scala> val inputfile = sc.textFile
("input.txt")
```

On executing the above command, the following output is observed -



element in the list, and flatte ; into the original list.

b. Each word is read and key-value pairs are created for each one of them using **map** transformation. This will assign the value '1' to each of the work-keys.

c. Finally the values of similar keys are added to get the final word count using **reduceByKey** function.

```
scala> val counts = inputfile.
flatMap (line => line. Split ("
")).map (word => (word,
1)).reduceByKey (_+_)
```

You will get the following output:

```
scala> val counts = inputfile.flatMap(line => line.split(" ")).map(word => (word, 1))
counts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <c
```

The next step is to store the output in a text file and exit the spark shell.

```
scala>counts.saveAsTextFile
("output")
```

```
scala> counts.saveAsTextFile("output")

scala> exit
```

Go to the output directory (location where you have created the file named output). Use 'ls'

REQUEST INFO

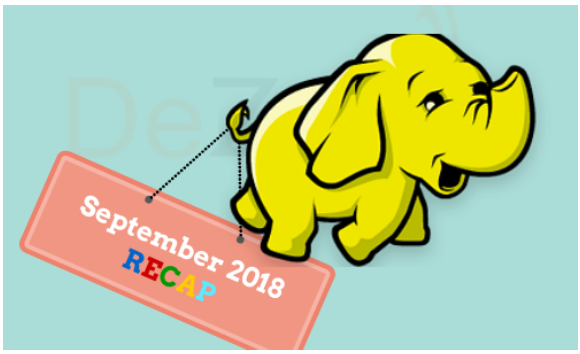
Using the cat command, print the contents of the output file to find the occurrence of each word in the input.txt file -

```
himanshu@himanshu-VirtualBox:~$ cd output
himanshu@himanshu-VirtualBox:~/output$ ls
part-00000  _SUCCESS
himanshu@himanshu-VirtualBox:~/output$ cat
(fail,1)
(Led,1)
(next,1)
(nor,1)
(rose,1)
(smallness,1)
(countryman,1)
(are.,1)
(Had,1)
(here,1)
(woody,1)
(put.,1)
```

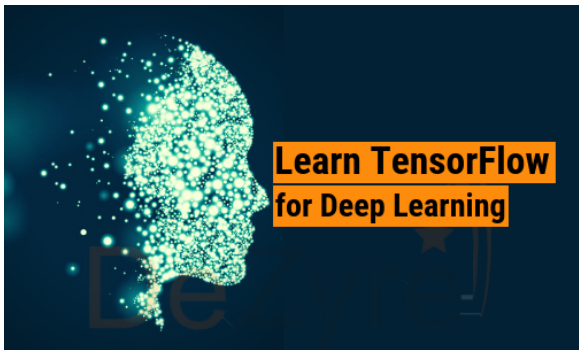
PREVIOUS NEXT

Data Cleaning in Python

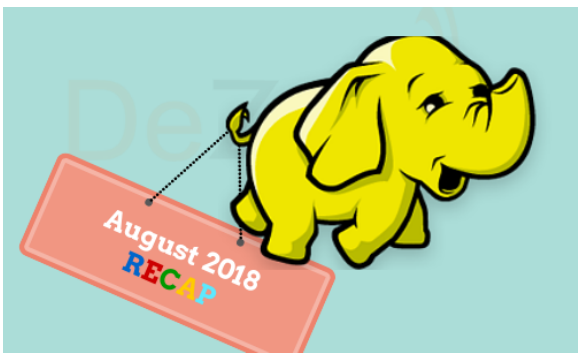
Python Pandas Dataframe Tutorials



Recap of Hadoop News for September 2018



Introduction to TensorFlow for Deep Learning



Recap of Hadoop News for August 2018

AWS vs Azure-Who is the big winner in the cloud war?

Other Tutorials

Big Data and Hadoop Training Courses in Popular Cities

- Microsoft Big Data and Hadoop Certification
- Hadoop Training in Texas
- Hadoop Training in California
- Hadoop Training in Dallas
- Hadoop Training in Chicago
- Hadoop Training in Charlotte
- Hadoop Training in Dubai
- Hadoop Training in Edison
- Hadoop Training in Fremont
- Hadoop Training in San Jose
- Hadoop Training in New Jersey
- Hadoop Training in New York
- Hadoop Training in Atlanta
- Hadoop Training in Canada
- Hadoop Training in Abu Dhabi
- Hadoop Training in Detroit
- Hadoop Trainging in Germany
- Hadoop Training in Houston
- Hadoop Training in Virginia
- Hadoop Training in Washington