Iceland Liechtenstein Norway grants

Computing Innovation for Technology Entrepreneurship

Information and Communications Technology based Innovation

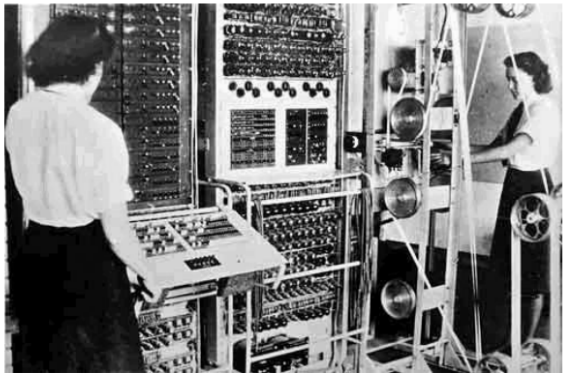
CITE
Computing Innovation for Technology Entrepreneurship

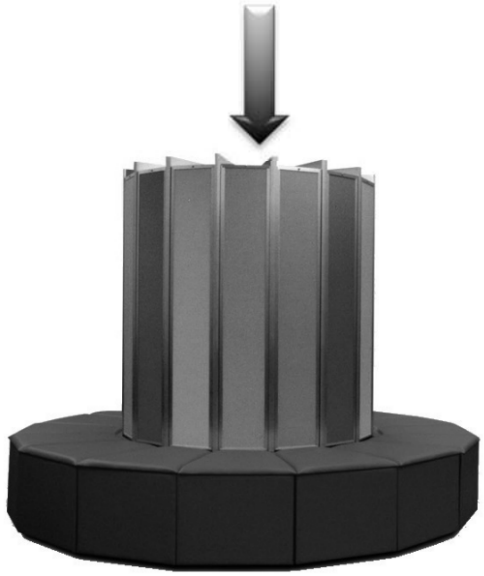
Technologies for Big Data Analysis

PhD. Eng. Iuliana MARIN

Why is Big Data Needed?

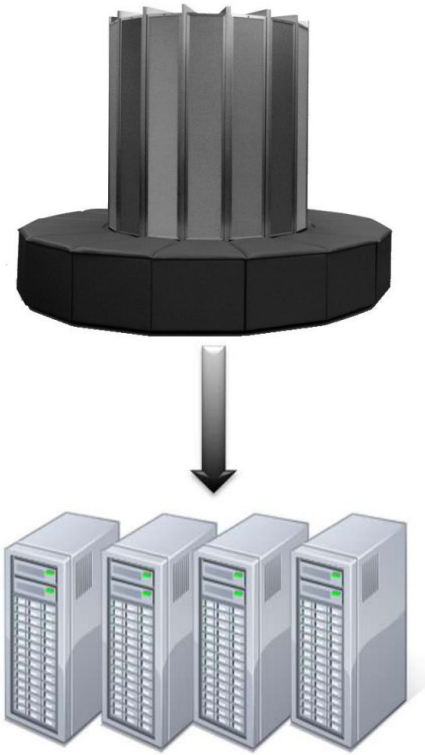
- Traditionally, computation has been processor-bound
  - Relatively small amounts of data
  - A lot of complex processing
- An early solution: bigger computers
  - Faster processor, more memory
  - But it still could not keep up





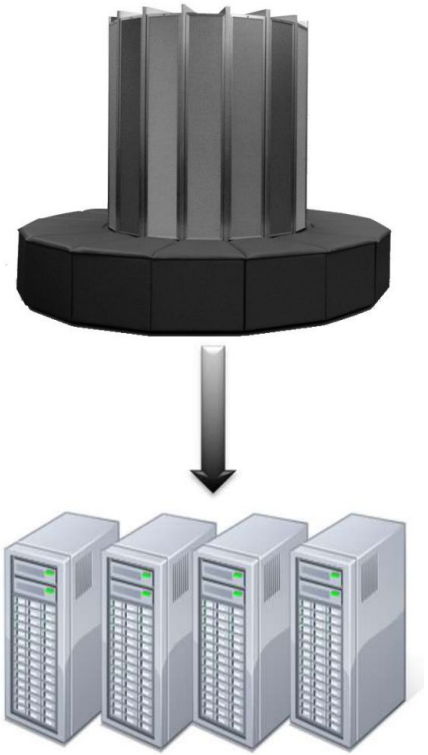
Why is Big Data Needed?

- A better solution: more computers
  - Distributed systems evolved
  - Use multiple machines for a single job
  - MPI (Message Passing Interface) as an example



Why is Big Data Needed?

- Distributed Systems Problems
  - Programming for traditional distributed systems is complex
    - Data exchange requires synchronization
    - Finite bandwidth is available
    - Temporal dependencies are complicated
    - It is difficult to deal with partial failures of the system
  - Ken Arnold, CORBA designer:
    - "Failure is the defining difference between distributed and local programming, so you have to design distributed systems with the expectation of failure"
    - Developers spend more time designing for failure than they do actually working on the problem itself

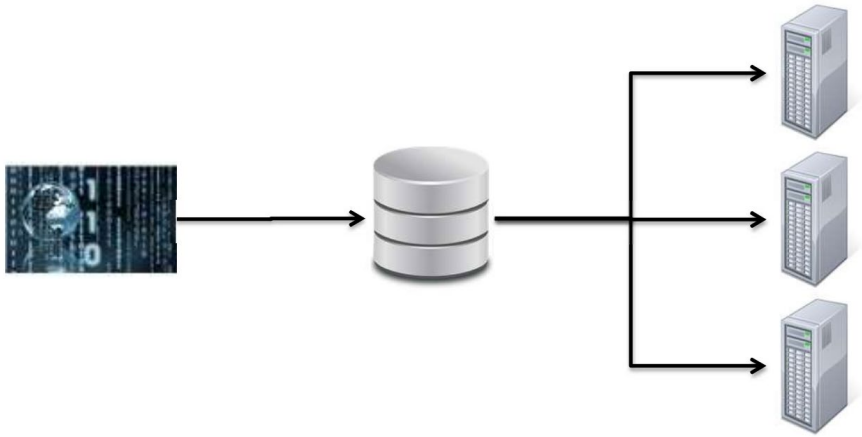


Why is Big Data Needed?

- Moore's Law has held firm for over 40 years
  - Processing power doubles every two years
  - Processing speed is no longer the problem
- Getting the data to the processors becomes the bottleneck
- Quick calculation
  - Typical disk data transfer rate: 75MB/sec
  - Time taken to transfer 100GB of data to the processor: approximately 22 minutes!
    - Assuming sustained reads
    - Actual time will be worse, since most servers have less than 100GB of RAM available
  - A new approach is needed

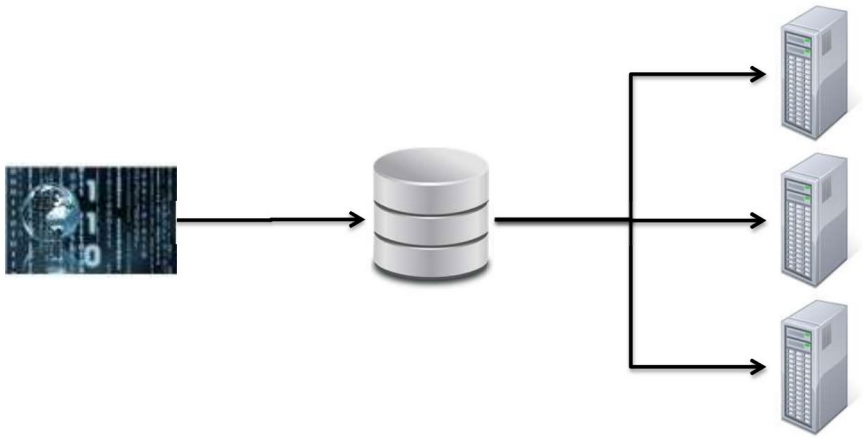
Distributed Systems: The Data Bottleneck

- Traditionally, data is stored in a central location
- Data is copied to processors at runtime
- Fine for limited amounts of data
- Modern systems have much more data
  - terabytes+ per day
  - petabytes+ total
- We need a new approach…



Partial Failure Support

- The system must support partial failure
  - Failure of a component should result in a graceful degradation of application performance
  - Not complete failure of the entire system



Quiz

- A given server has a 1 in 1000 chance of failure on any given day, but you have 500 servers like this in your data center. What is the likely failure of your servers? (choose the best answer.)
  - a) Once in ten days
  - b) Once every two days
  - c) Once every day

Quiz

- A given server has a 1 in 1000 chance of failure on any given day, but you have 500 servers like this in your data center. What is the likely failure of your servers? (choose the best answer.)
  - a) Once in ten days
  - b) Once every two days
  - c) Once every day

- Answer: b. If a given server has a 1 in 1000 chance of failure on any given day, but you have 100 servers like this in your data center, you will likely have a failure once every ten days. And when you "scale out" to 500 servers, the problem actually becomes worse because you'll have a failure every two days.

Data Recoverability

- If a component of the system fails, its workload should be assumed by still-functioning units in the system
  - Failure should not result in the loss of any data
- If a component of the system fails and then recovers, it should be able to rejoin the system
  - Without requiring a full restart of the entire system
- **Consistency** - Component failures during execution of a job should not affect the outcome of the job
- **Scalability** - Adding load to the system should result in a graceful decline in performance of individual jobs
  - Not failure of the system
  - Increasing resources should support a proportional increase in load capacity

Hadoop's History

- Hadoop is based on work done by Google in the late 1990s/early 2000s
  - Specifically, on papers describing the Google File System (GFS) published in 2003, and MapReduce published in 2004
  - This work takes a radical new approach to the problem of distributed computing
    - Meets all the requirements we have for reliability and scalability
  - Core concept: distribute the data as it is initially stored in the system Individual nodes can work on data local to those nodes
    - No data transfer over the network is required for initial processing

Core Hadoop Concepts

- Applications are written in high-level code
  - Developers need not worry about network programming, temporal dependencies or low-level infrastructure
- Nodes talk to each other as little as possible
  - Developers should not write code which communicates between nodes
  - 'Shared nothing' architecture
- Data is spread among machines in advance
  - Computation happens where the data is stored, wherever possible
    - Data is replicated multiple times on the system for increased availability and reliability

Hadoop: Very High-Level Overview

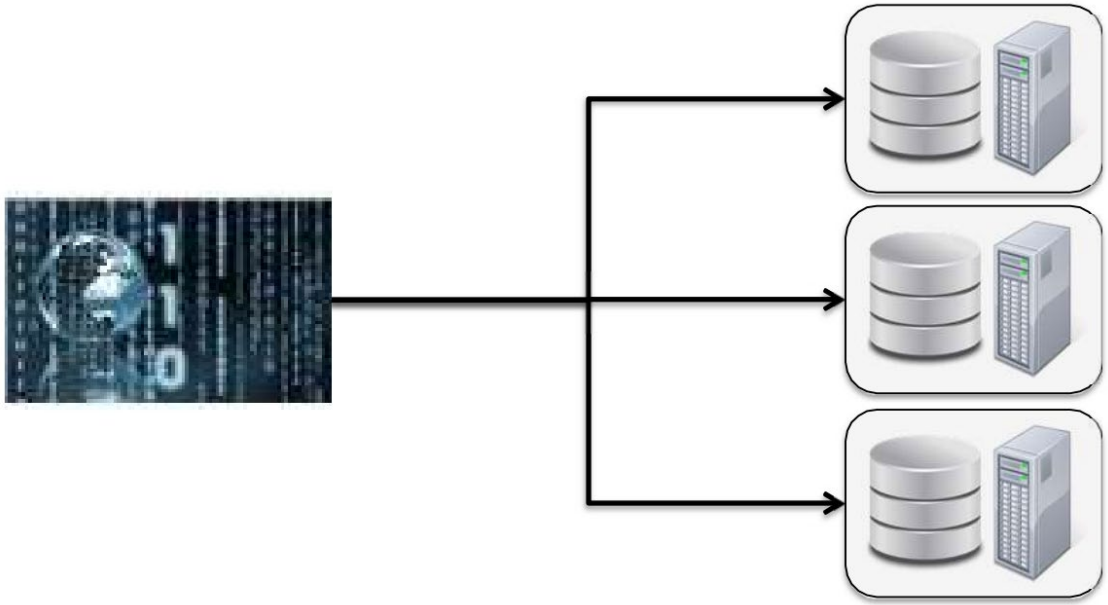
- When data is loaded into the system, it is split into 'blocks'
  - Typically 64MB or 128MB
- Map tasks (the first part of the MapReduce system) work on relatively small portions of data
  - Typically a single block
- A master program allocates work to nodes such that a Map task will work on a block of data stored locally on that node whenever possible
  - Many nodes work in parallel, each on their own part of the overall dataset

Fault Tolerance

- If a node fails, the master will detect that failure and re-assign the work to a different node on the system
- Restarting a task does not require communication with nodes working on other portions of the data
- If a failed node restarts, it is automatically added back to the system and assigned new tasks
- If a node appears to be running slowly, the master can redundantly execute another instance of the same task
  - Results from the first to finish will be used
  - Known as 'speculative execution'

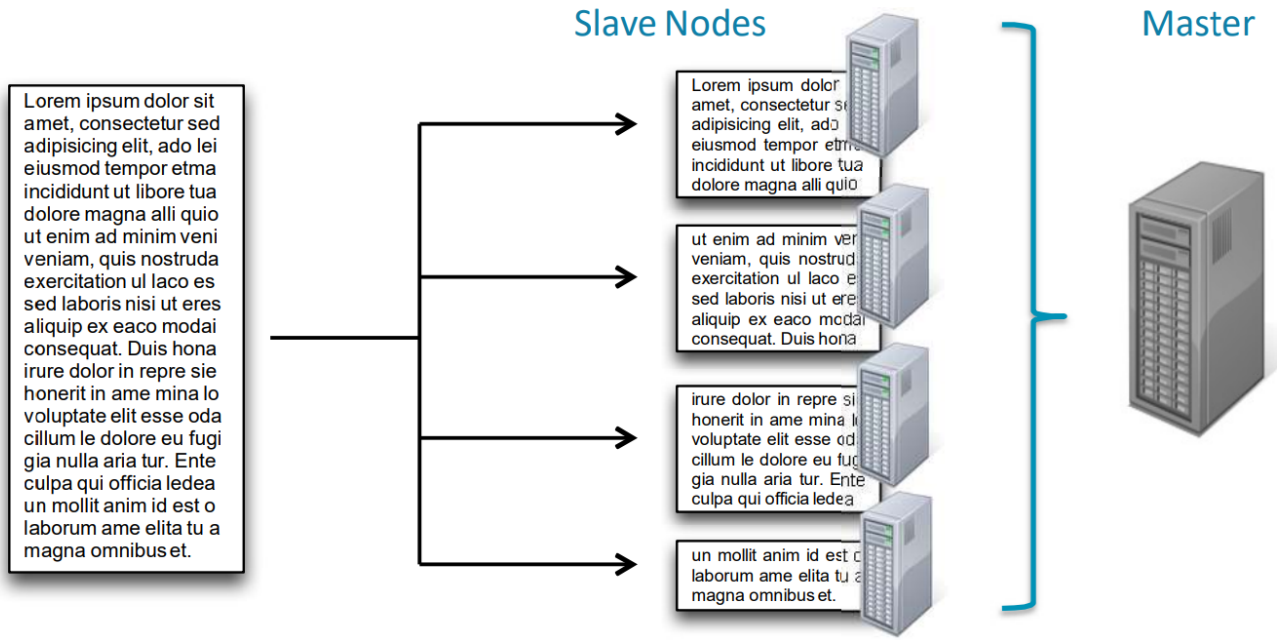
Hadoop

- A radical new approach to distributed computing
  - Distribute data when the data is being stored
  - Run computation where the data is stored



Hadoop: Very High-Level Overview

- Data is split into "blocks" when loaded
- Map tasks typically work on a single block
- A master program manages tasks



Core Hadoop Concepts

- Applications are written in high-level code
- Nodes talk to each other as little as possible
- Data is distributed in advance
  - Bring the computation to the data
- Data is replicated for increased availability and reliability
- Hadoop is scalable and fault-tolerant

Quiz

- If "mean time to failure" for one disk is 3 years = about 1000 days, but you have 100 computers in a cluster with 10 disks per computer, what is the likely disk failure in your cluster? (choose the best answer.)
  - a) Once in ten days
  - b) Once every two days
  - c) Once every day

Quiz

- If "mean time to failure" for one disk is 3 years = about 1000 days, but you have 100 computers in a cluster with 10 disks per computer, what is the likely disk failure in your cluster? (choose the best answer.)
  - a) Once in ten days
  - b) Once every two days
  - c) Once every day

- Answer: c.

100 computers in a cluster

10 disks per computer = 1000 disks

If "mean time to failure" for one disk is 3 years = about 1000 days.

For 1000 disks, Disk failure on average happens once per day.

Iceland Liechtenstein Norway grants

Computing Innovation for Technology Entrepreneurship

Information and Communications Technology based Innovation

CITE
Computing Innovation for Technology Entrepreneurship

Big data analysis with Scala and Spark

Apache Spark

- Apache Spark is an Open source analytical processing engine for large scale powerful distributed data processing and machine learning applications.
- Spark is Originally developed at the University of California, Berkeley's, and later donated to Apache Software Foundation.
- In February 2014, Spark became a Top-Level Apache Project and has been contributed by thousands of engineers and made Spark as one of the most active open-source projects in Apache.

Apache Spark

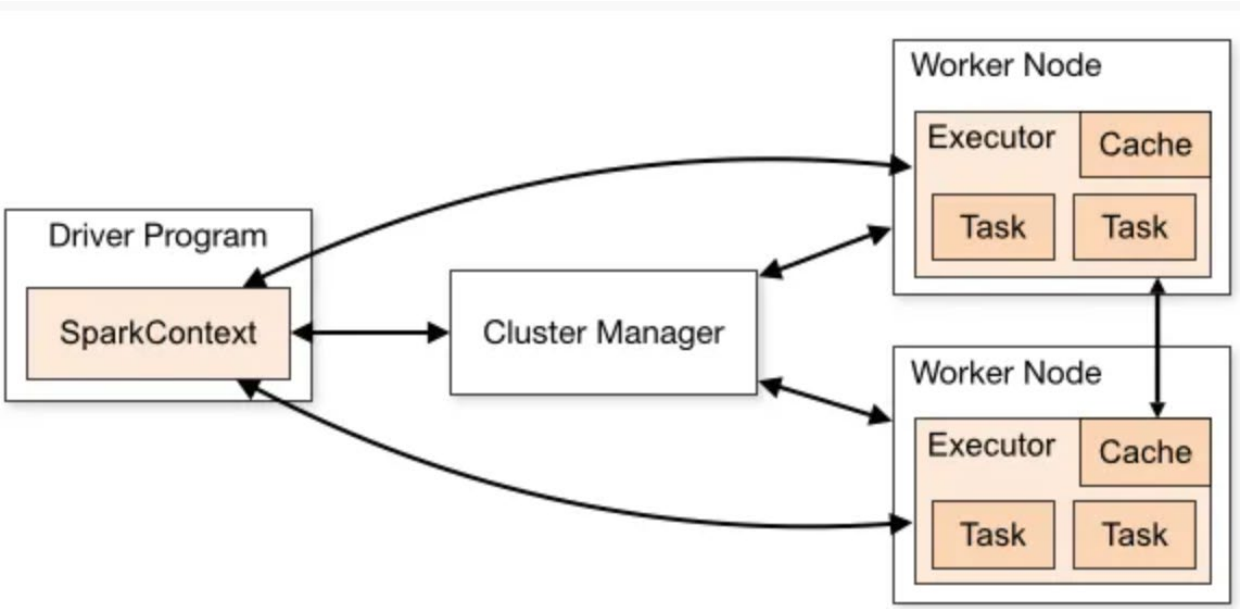
- In-memory computation
- Distributed processing using parallelize
- Can be used with many cluster managers (Spark, Yarn, Mesos)
- Fault-tolerant
- Immutable
- Lazy evaluation
- Cache & persistence
- Inbuild-optimization when using DataFrames
- Supports ANSI SQL

Apache Spark Advantages

- Spark is a general-purpose, in-memory, fault-tolerant, distributed processing engine that allows you to process data efficiently in a distributed fashion.
- Applications running on Spark are 100x faster than traditional systems.
- You will get great benefits using Spark for data ingestion pipelines.
- Using Spark we can process data from Hadoop HDFS, AWS S3, Databricks DBFS, Azure Blob Storage, and many file systems.
- Spark also is used to process real-time data using Streaming and Kafka.
- Using Spark Streaming you can also stream files from the file system and also stream from the socket.
- Spark natively has machine learning and graph libraries.

Apache Spark Architecture

- Apache Spark works in a master-slave architecture where the master is called “Driver” and slaves are called “Workers”. When you run a Spark application, Spark Driver creates a context that is an entry point to your application, and all operations (transformations and actions) are executed on worker nodes, and the resources are managed by Cluster Manager.

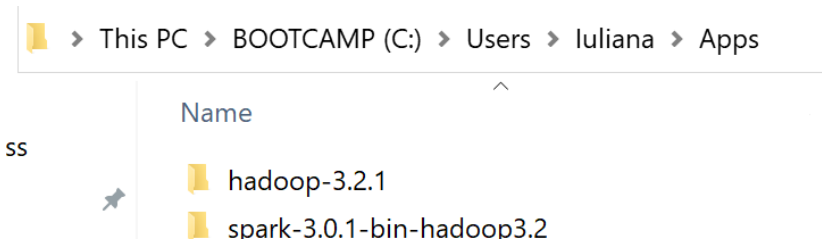


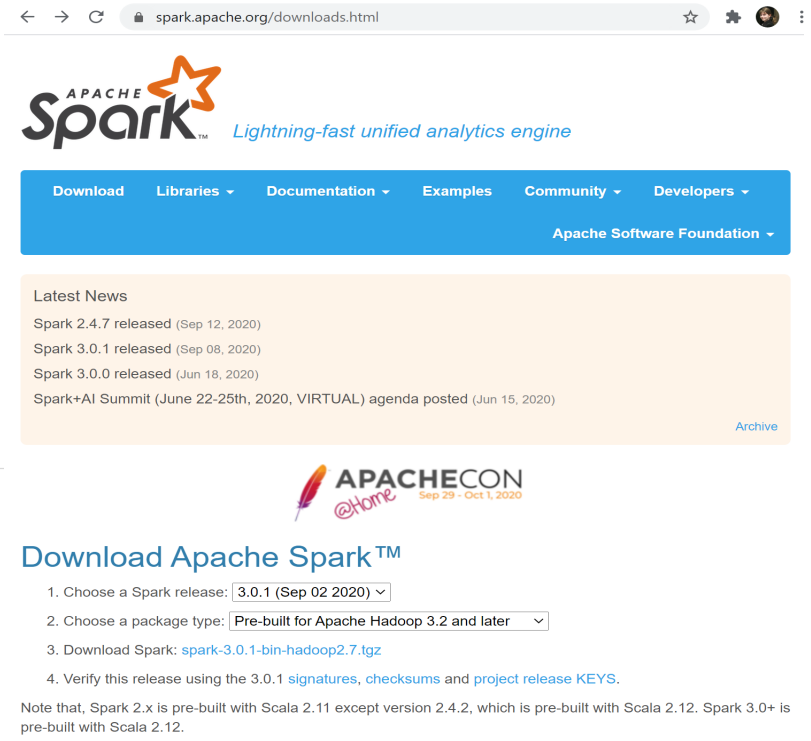
Cluster Manager Types

- Standalone – a simple cluster manager included with Spark that makes it easy to set up a cluster.
- Apache Mesos – Mesons is a Cluster manager that can also run Hadoop MapReduce and Spark applications.
- Hadoop YARN – the resource manager in Hadoop 2. This is mostly used, cluster manager.
- Kubernetes – an open-source system for automating deployment, scaling, and management of containerized applications.

Spark Installation

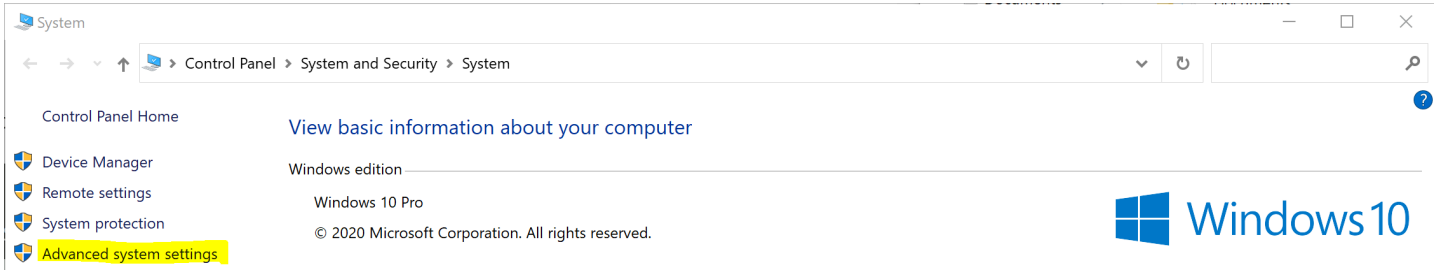
- Download Apache Spark by accessing Spark Download page and select the link from "Download Spark (point 3)".
- After download, untar the binary using 7zip and copy the underlying folder spark-3.0.1-bin-hadoop3.2 to C:\Users\Iuliana\Apps.
- Install Java JDK 8.



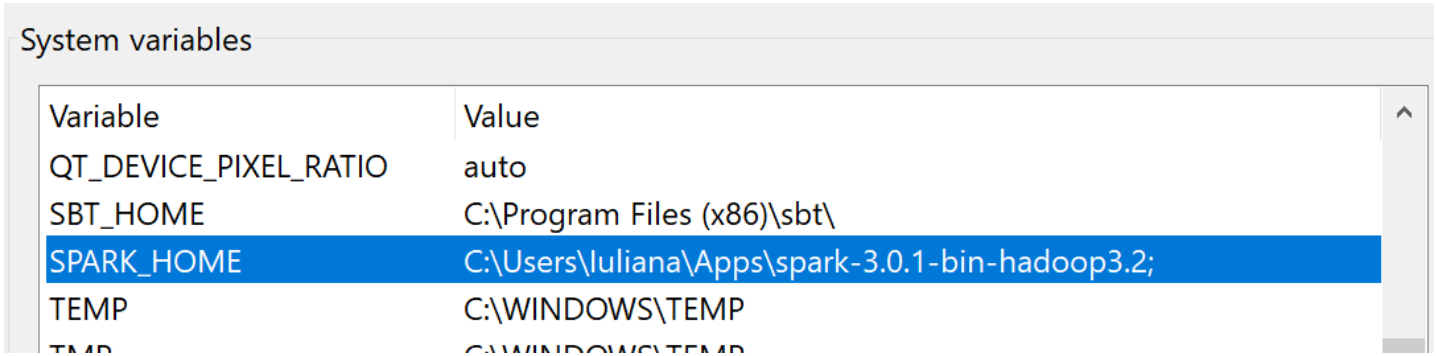


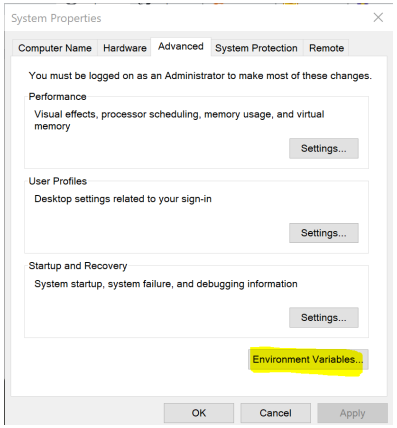
Spark Installation

- You can set the environment variables by going to This PC, right click, selecting Properties and then by clicking on Advanced system settings.



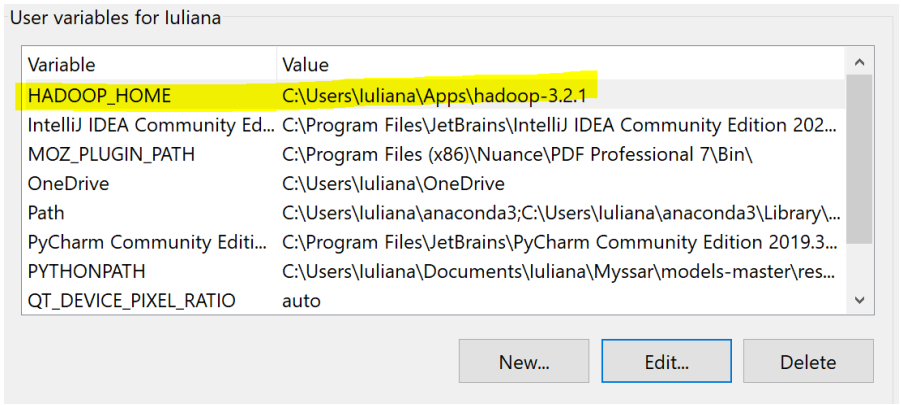
- Click on Environment Variables.
- For System variables, click on New and then add.





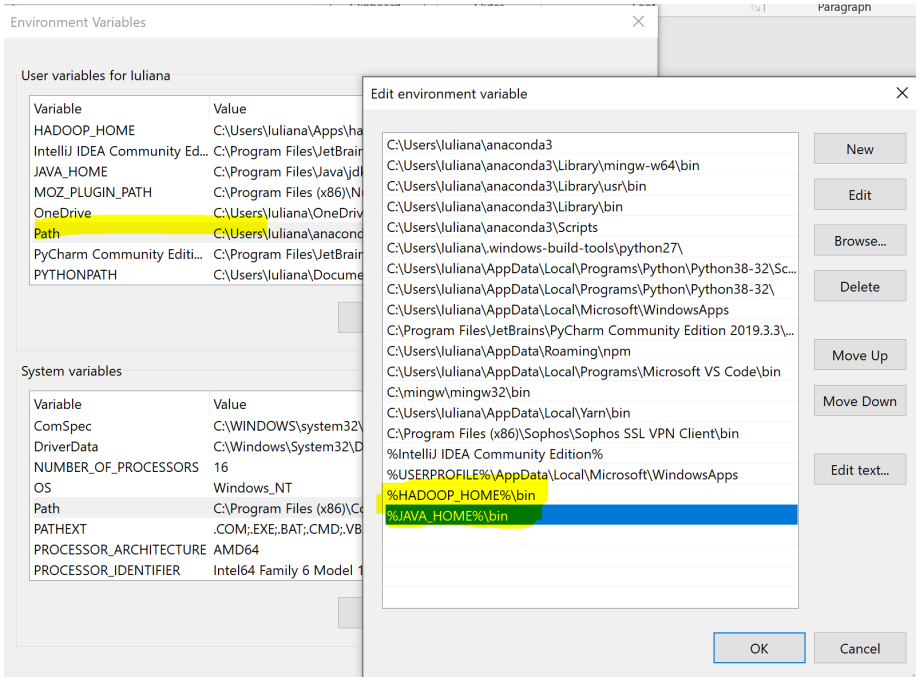
Spark Installation

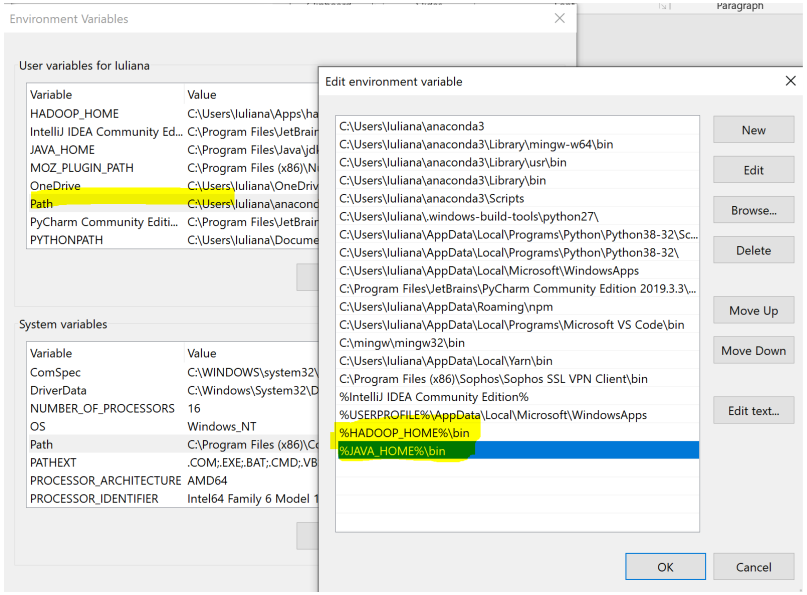
- Install JDK 8 from Oracle. Download Hadoop 3.2.1 from https://github.com/cdarlint/winutils/tree/master/hadoop-3.2.1.
- Unzip it inside the C:\Users\Iuliana\Apps folder. Add the libwinutils.lib, winutils.pdb and winutils files inside the bin folder. Winutils contain the Windows binaries for Hadoop versions.
- Set the environment variable for it.



Spark Installation

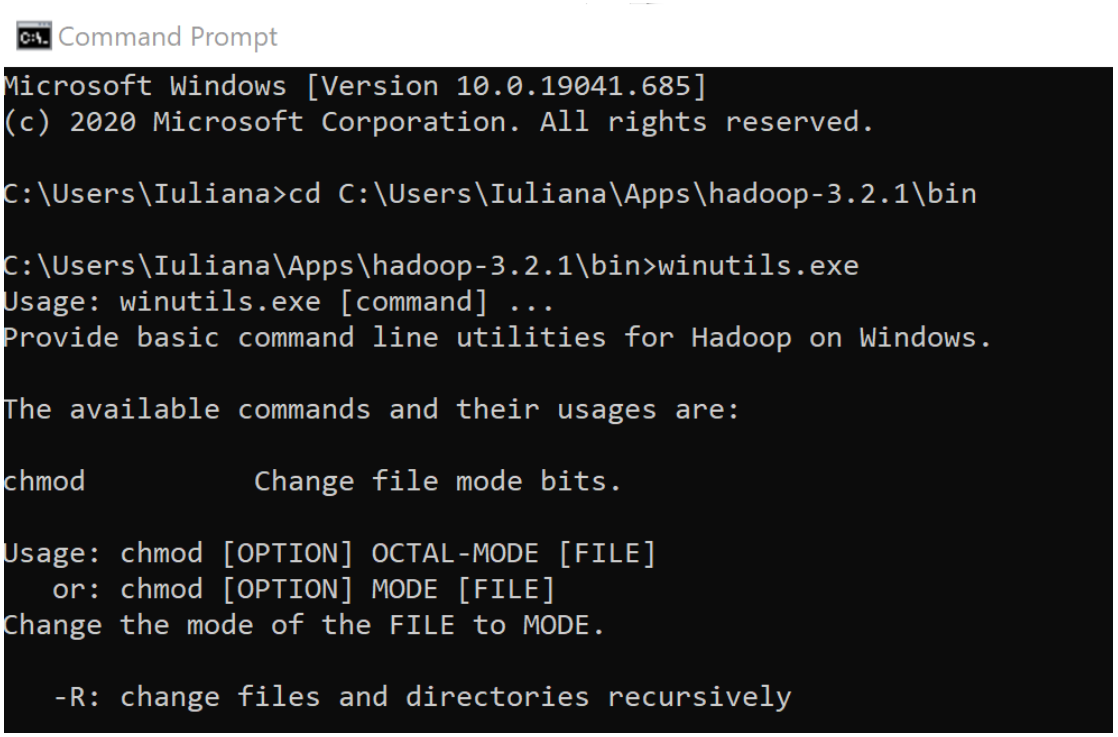
- Edit the Path and add the path towards the bin folders for Hadoop and Java:





Spark Installation

- Go to the path where Hadoop was installed and then run winutils.exe.

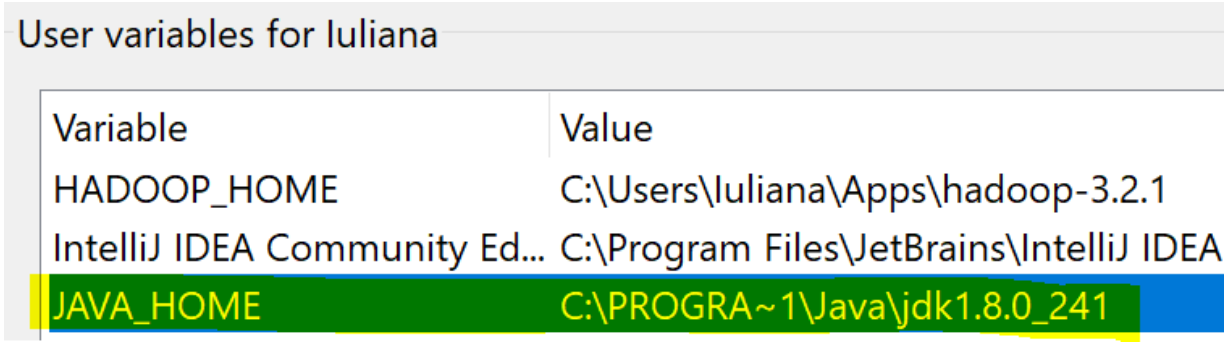


Spark Installation

- Test the Hadoop version using the command "hadoop –version".

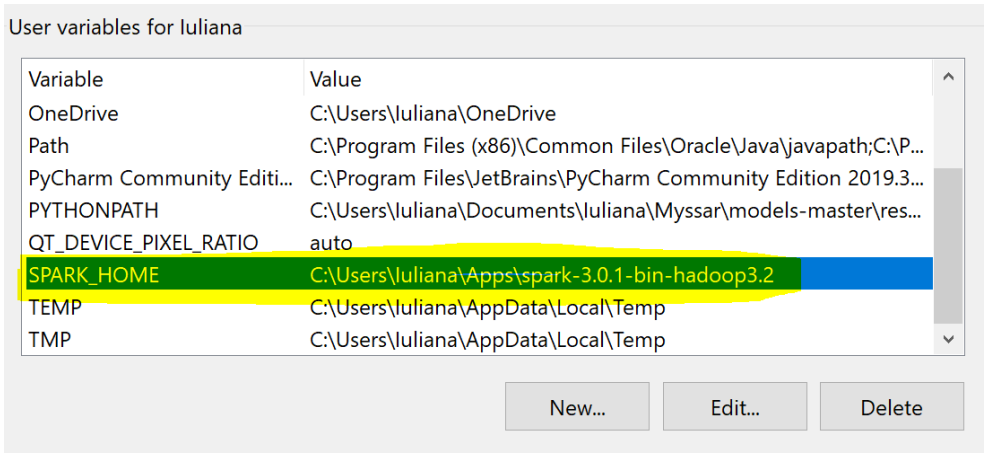
```
C:\Users\Iuliana\Apps\hadoop-3.2.1\bin>hadoop -version
java version "1.8.0_241"
Java(TM) SE Runtime Environment (build 1.8.0_241-b07)
Java HotSpot(TM) 64-Bit Server VM (build 25.241-b07, mixed mode)
```

- If an error naming that JAVA_HOME was not found, then change its path into (by modifying Program Files with PROGRA~1):

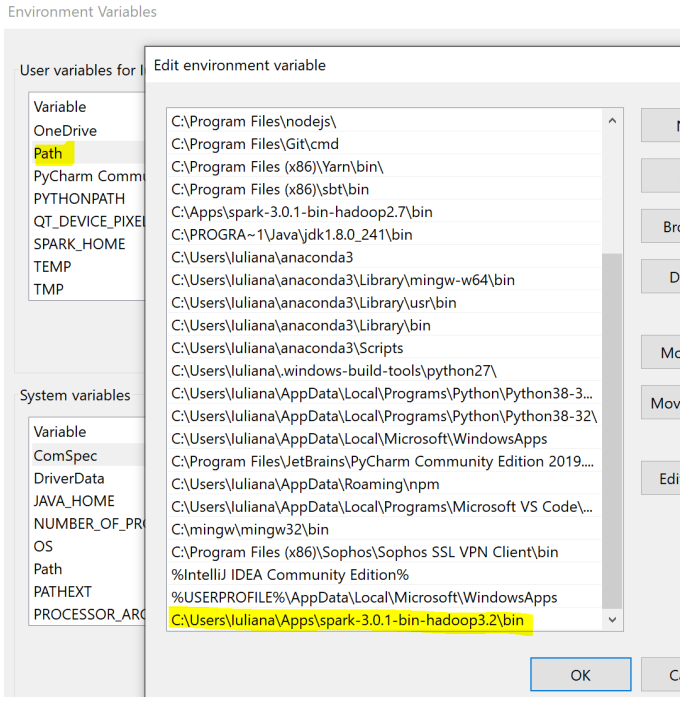


Spark Installation

- Set SPARK_HOME



- Add Spark inside Path.



Spark Installation

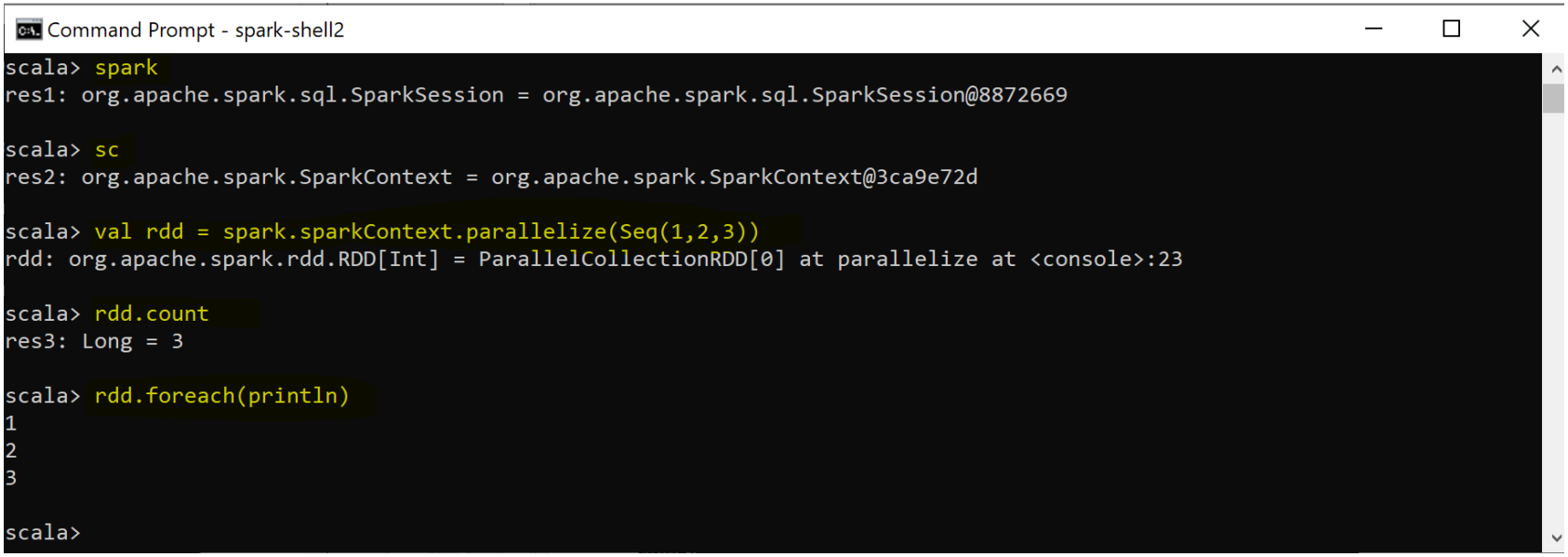
- Start Spark shell, by going to its folder and run the command “spark-shell2”.

Command Prompt - spark-shell2

```
Microsoft Windows [Version 10.0.19041.685]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Iuliana>cd C:\Users\Iuliana\Apps\spark-3.0.1-bin-hadoop3.2

C:\Users\Iuliana\Apps\spark-3.0.1-bin-hadoop3.2>cd bin

C:\Users\Iuliana\Apps\spark-3.0.1-bin-hadoop3.2\bin>spark-shell2
20/12/16 20:11:10 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://192.168.0.220:4040
Spark context available as 'sc' (master = local[*], app id = local-1608142275687).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.0.1
      /_/

Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_241)
Type in expressions to have them evaluated.
Type :help for more information.
```

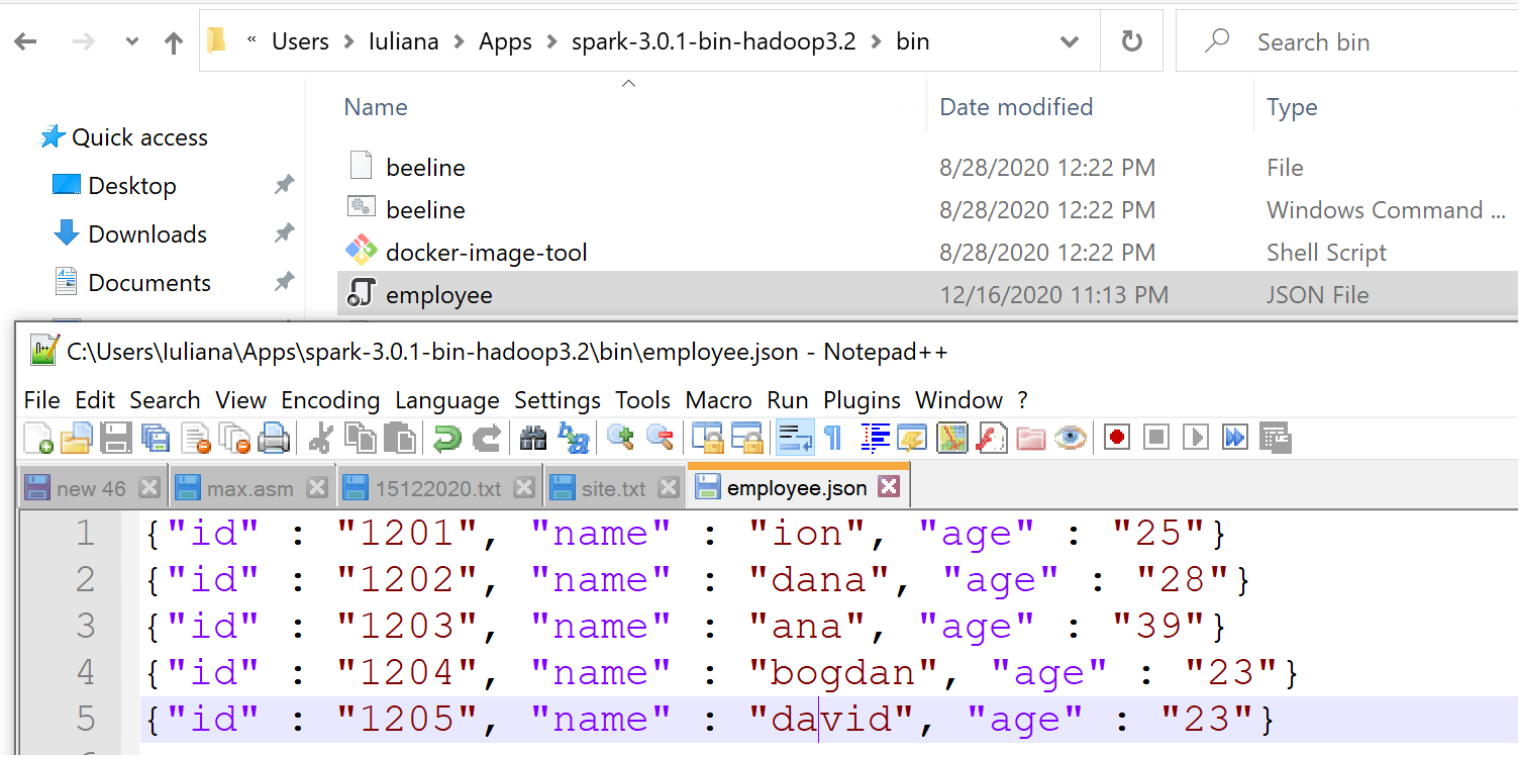
Spark Installation

- Press enter inside the Command Prompt and run the following commands to test the environment's functionalities.



Spark and JSON Files

- Create a JSON file inside the bin folder of Spark. The file will contain several employees.



SQL Management

- Use the following command to create the SQLContext. Use the following command to read the JSON document named employee.json.

```
scala> val sqlContext = new org.apache.spark.sql.SQLContext(sc)
```

- The data is shown as a table with the fields – id, name, and age.

```
scala> val dfs = sqlContext.read.json("employee.json")
dfs: org.apache.spark.sql.DataFrame = [age: string, id: string ... 1 more field]

scala> dfs.show()
+---+----+------+
|age|  id|  name|
+---+----+------+
| 25|1201|   ion|
| 28|1202|  dana|
| 39|1203|   ana|
| 23|1204|bogdan|
| 23|1205| david|
+---+----+------+
```

SQL Management

- The structure (schema) of the DataFrame can be seen using the following command:

```
scala> dfs.printSchema()
root
 |-- age: string (nullable = true)
 |-- id: string (nullable = true)
 |-- name: string (nullable = true)
```

- Get the values of the name column which belong to the DataFrame.

```
scala> dfs.select("name").show()
+------+
|  name|
+------+
|   ion|
|  dana|
|   ana|
|bogdan|
| david|
+------+
```

SQL Management

- A filter can be applied based on age to find the employees who have the age greater than 24.

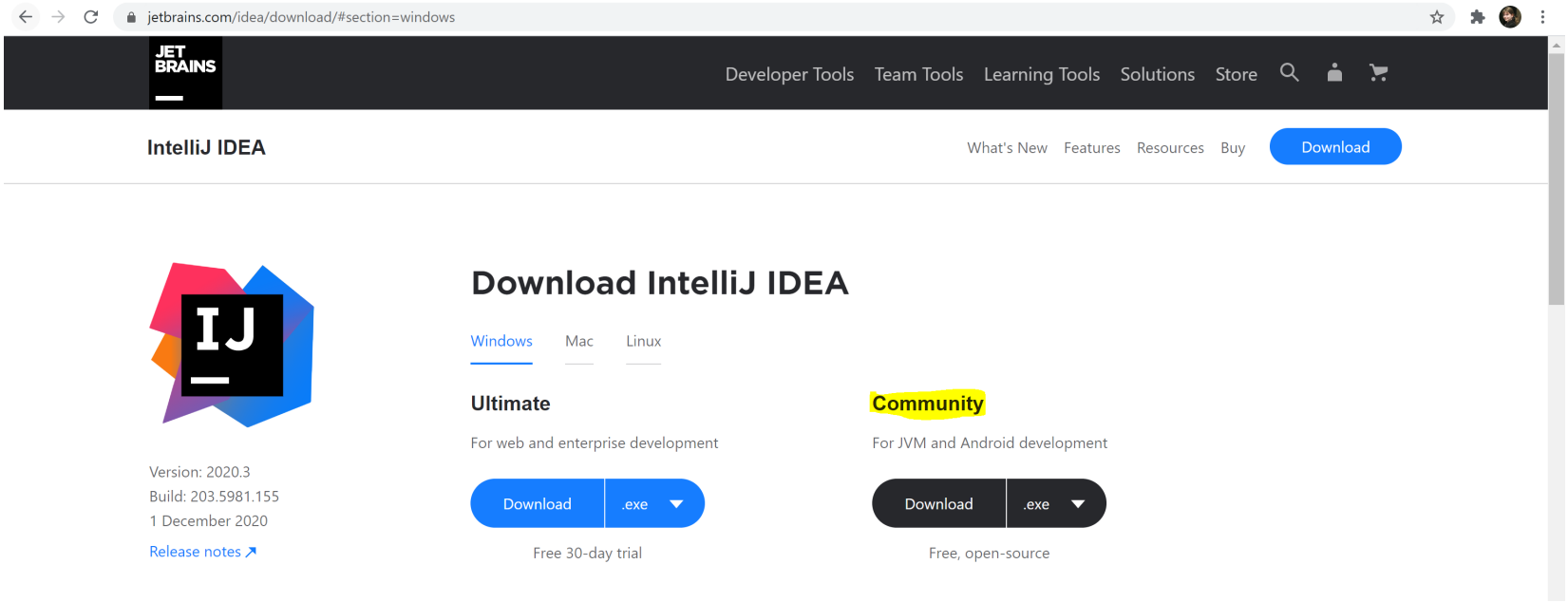
```
scala> dfs.filter(dfs("age") > 24).show()
+---+----+----+
|age|  id|name|
+---+----+----+
| 25|1201| ion|
| 28|1202|dana|
| 39|1203| ana|
+---+----+----+
```

- GroupBy is used to count the number of employees who have the same age.

```
scala> dfs.groupBy("age").count().show()
+---+-----+
|age|count|
+---+-----+
| 28|    1|
| 23|    2|
| 25|    1|
| 39|    1|
+---+-----+
```

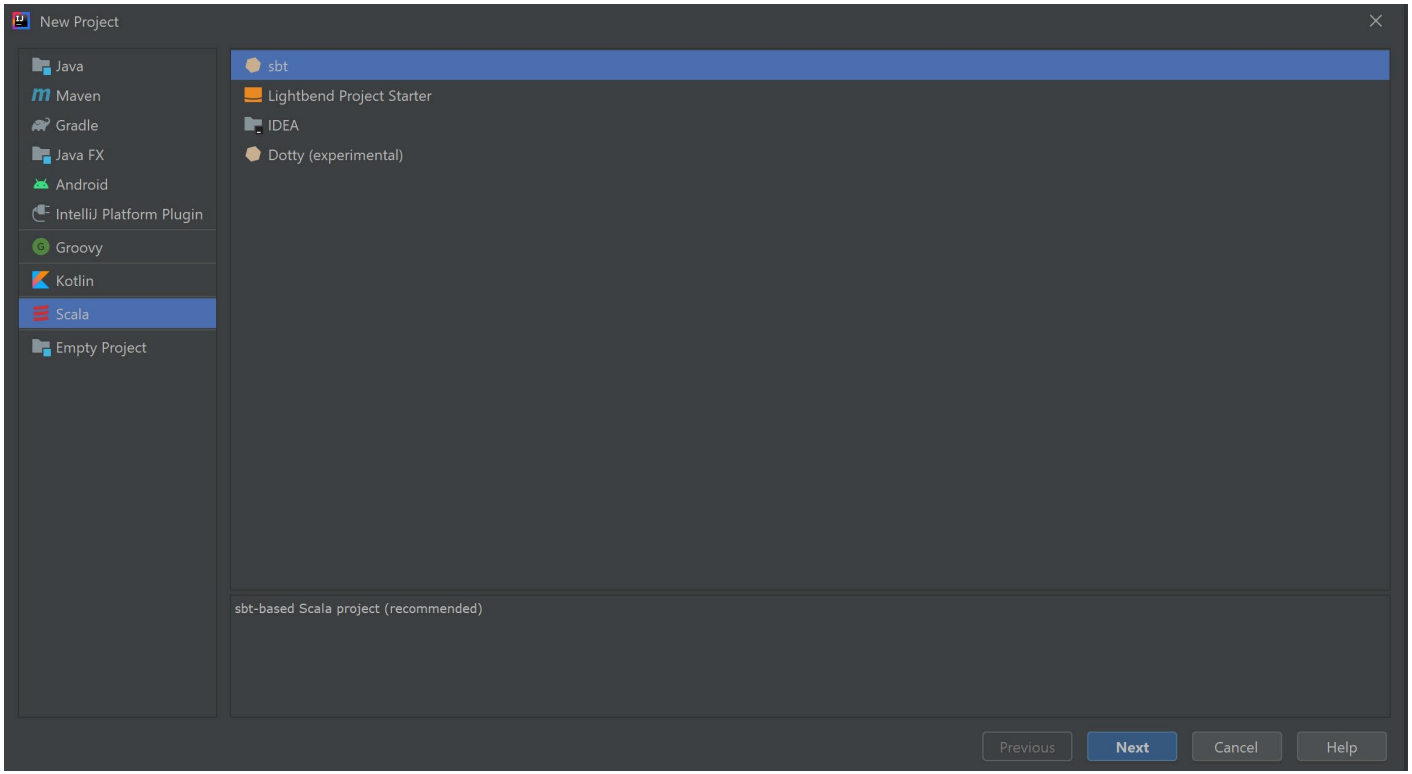
IntelliJ IDEA Community Installation

- Install IntelliJ IDEA Community edition to run Spark applications written in Scala due to it's good Scala compatibility.



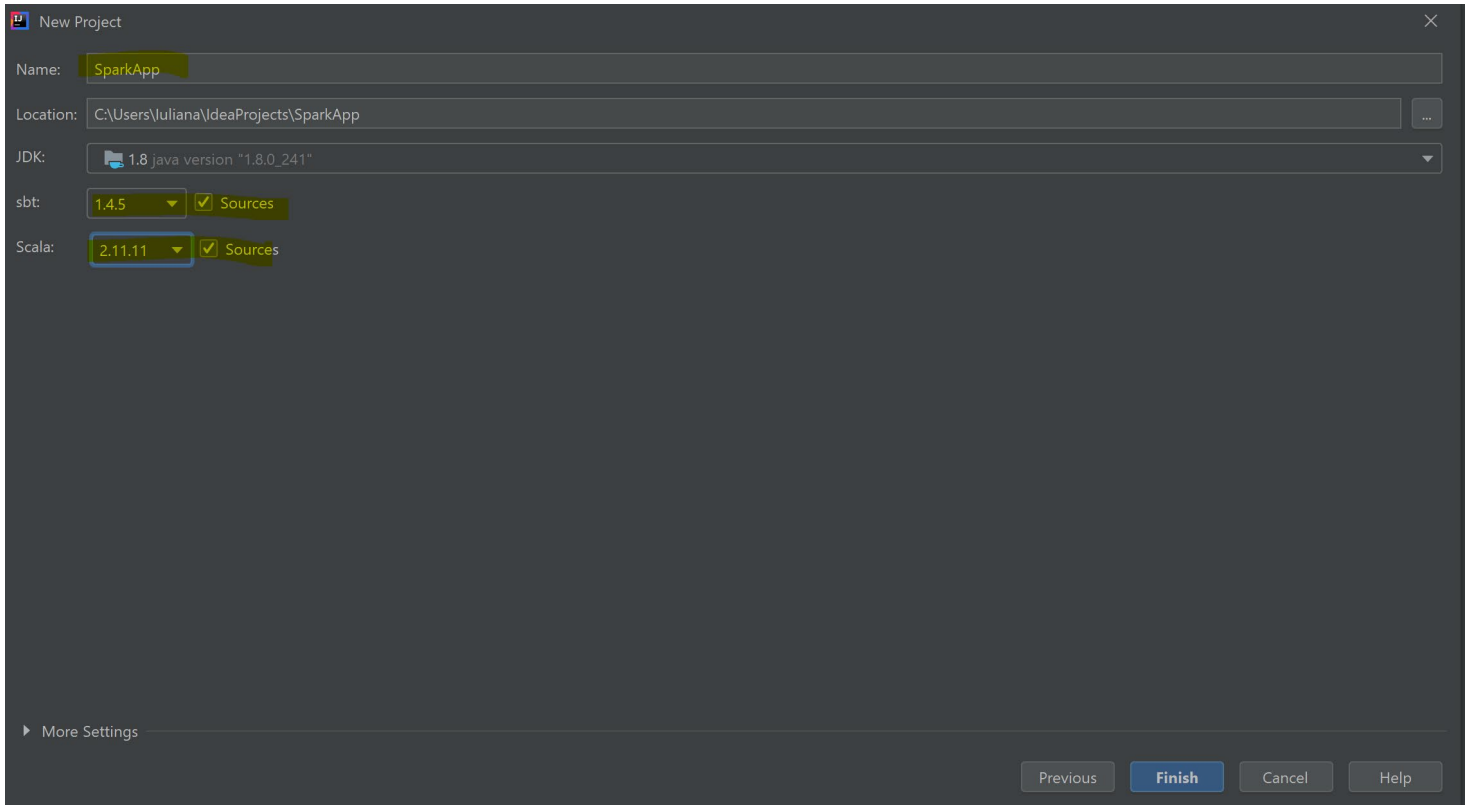
Spark Installation using SBT

- Install sbt using the official website.
- Create a new project by going to File -> New Project and the select Scala and sbt. Afterwards, click on Next.



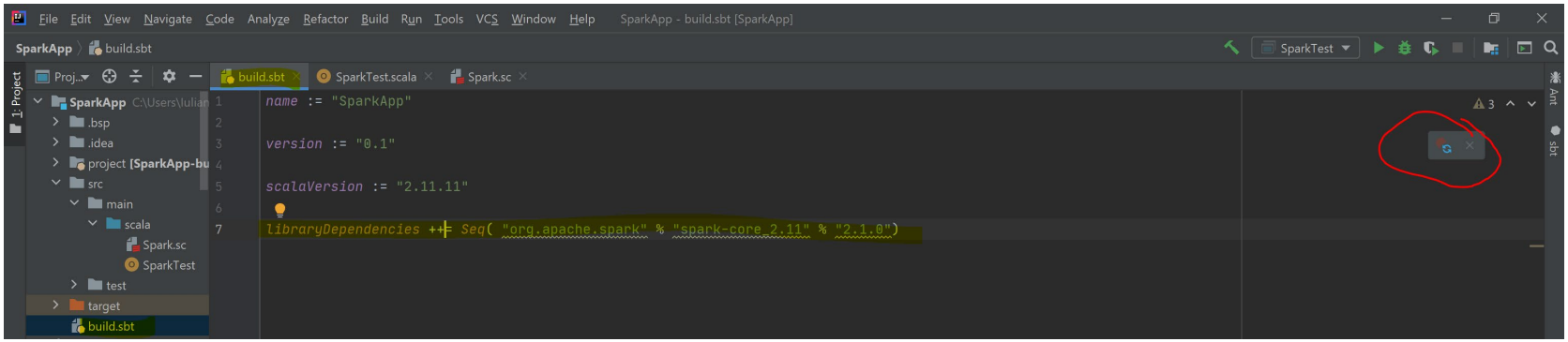
Spark Installation using SBT

- Type the name of the project and click the sbt checkbox for sources, as well as the one for Scala.



Spark Installation using SBT

- Open the build.sbt file and type the last row to include library dependencies for Spark.
- Click on the logo which is depicted in the picture with a red circle and let the program install the needed library.



MapReduce in Scala

- Open the src folder until you reach the scala folder. Right click on it, select New and choose Scala Worksheet. A file with the extension .sc will be created.
- Add the following MapReduce code which will display the top 10 words that appear in the copyright file of Java JDK.

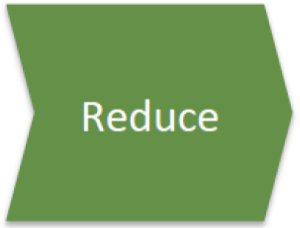
```
import org.apache.log4j.{Level, Logger}
import org.apache.spark.SparkContext

   Logger.getRootLogger.setLevel(Level.INFO)
   val sc = new SparkContext("local[*]" , "SparkDemo")
   val lines = sc.textFile("C:\\Program Files\\Java\\jdk1.8.0_241\\COPYRIGHT");
   val words = lines.flatMap(line => line.split(' '))
   val wordsKVRdd = words.map(x => (x,1))
   val count = wordsKVRdd.reduceByKey((x,y) => x + y).map(x =>
(x._2,x._1)).sortByKey(false).map(x => (x._2, x._1)).take(10)
   count.foreach(println)
```

Hadoop Components: MapReduce

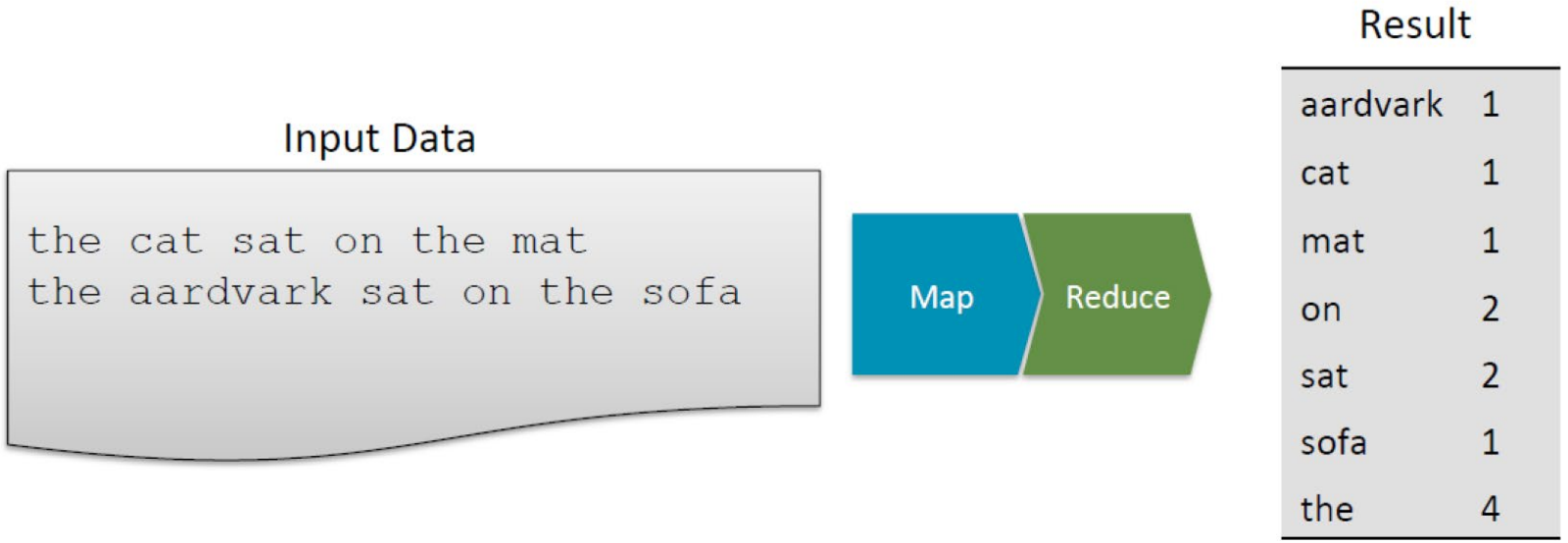
- **The Mapper**
  - Each Map task (typically) operates on a single HDFS block
  - Map tasks (usually) run on the node where the block is stored

- **Shuffle and Sort**
  - Sorts and consolidates intermediate data from all mappers
  - Happens as Map tasks complete and before Reduce tasks start

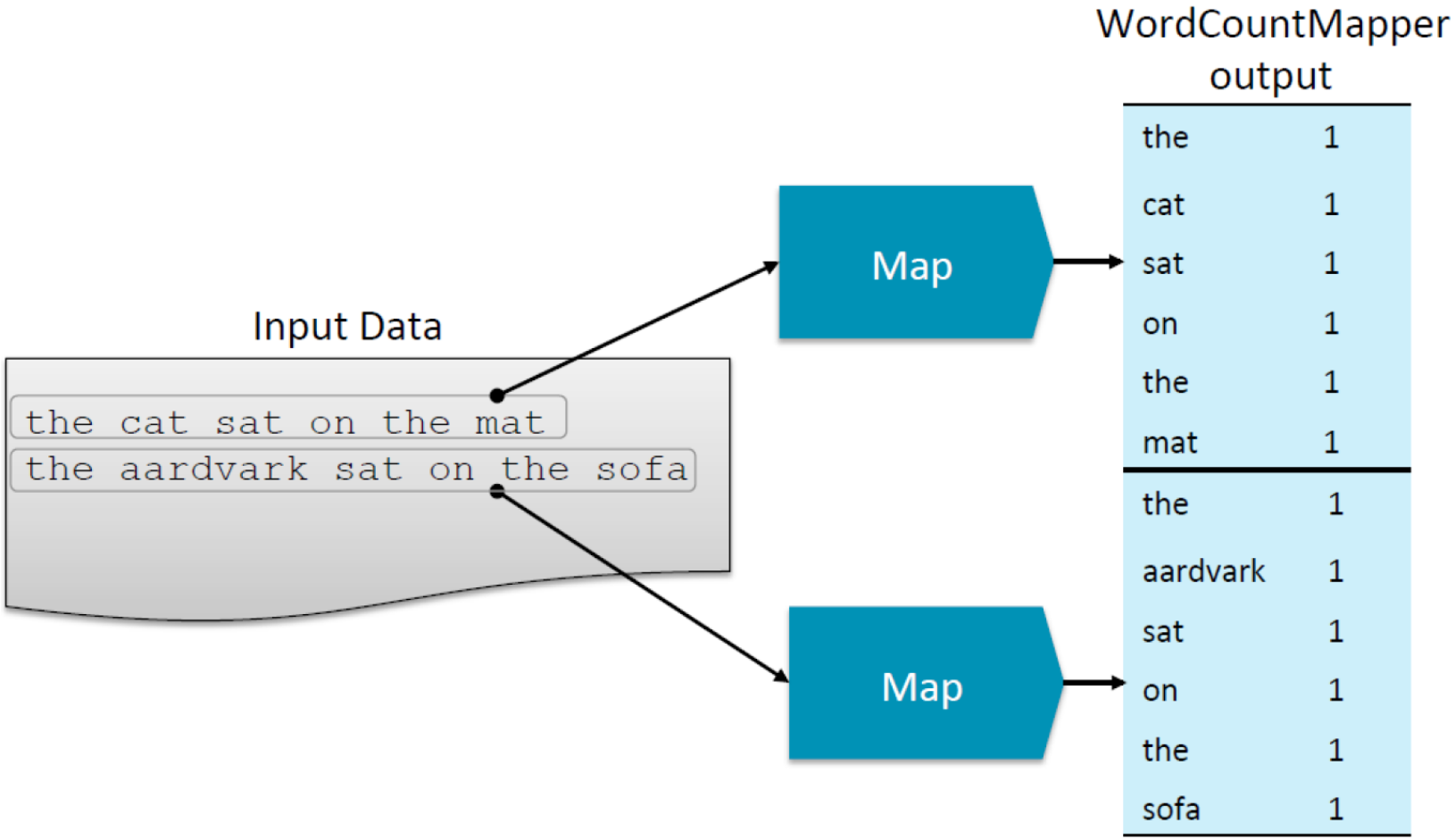
- **The Reducer**
  - Operates on shuffled/sorted intermediate data (Map task output)
  - Produces final output



Hadoop Components: MapReduce Example: Word Count



Example: Word Count Mapper



Example: Shuffle & Sort

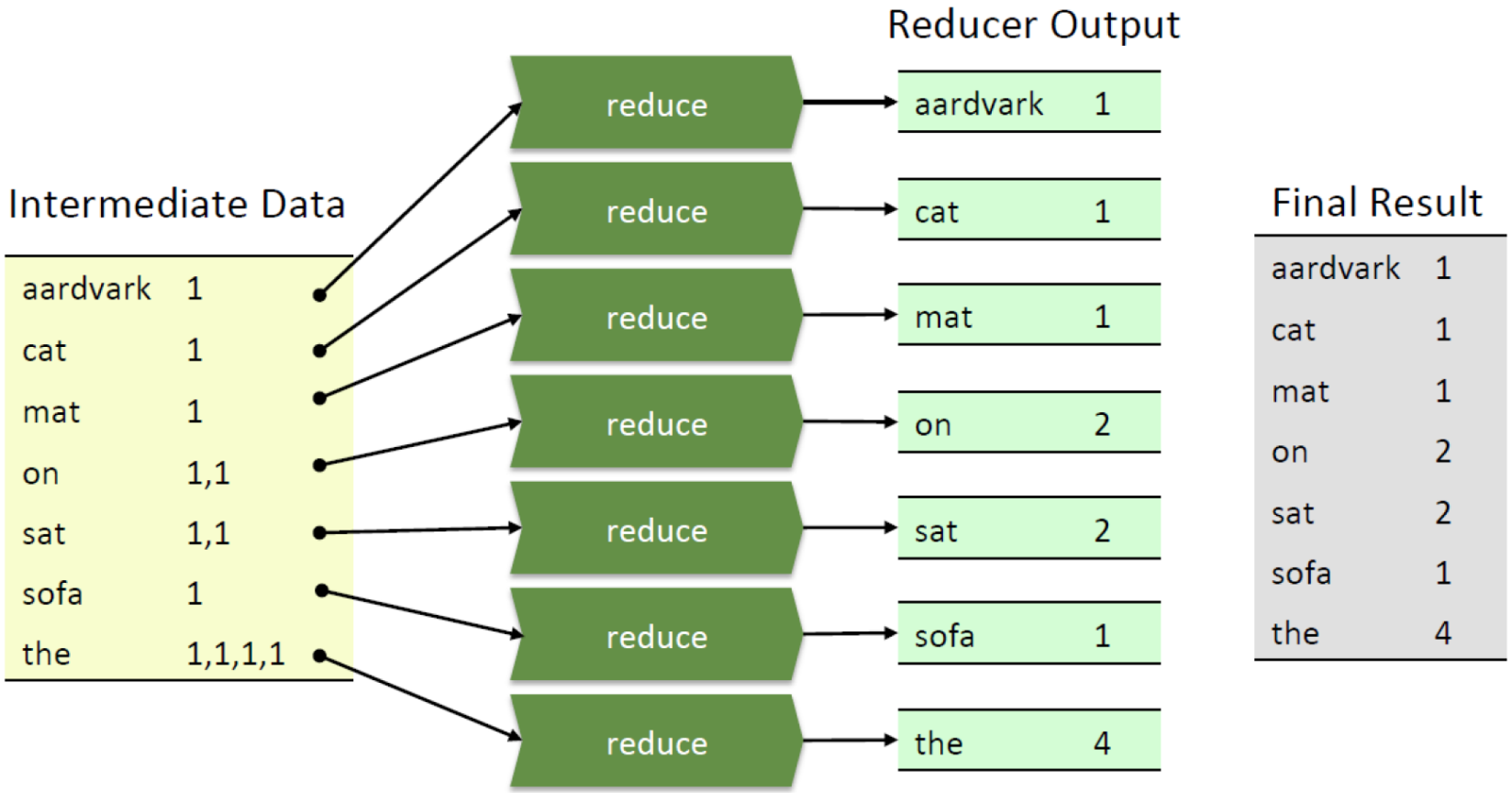
Mapper Output

the	1
cat	1
sat	1
on	1
the	1
mat	1
the	1
aardvark	1
sat	1
on	1
the	1
sofa	1

Intermediate Data

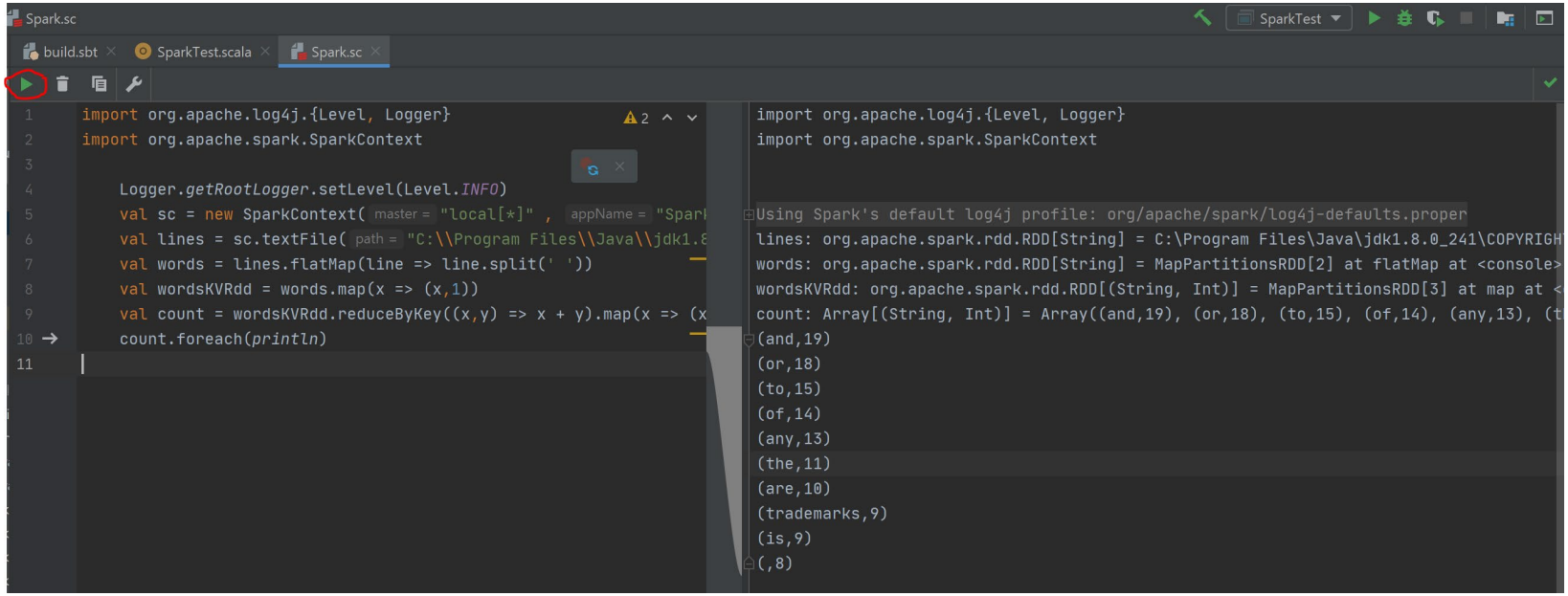
aardvark	1
cat	1
mat	1
on	1,1
sat	1,1
sofa	1
the	1,1,1,1

Example: SumReducer



MapReduce in Scala

- Click on the green Play button and the results will be displayed on the right side of the window.



References

- Google File System - https://research.google/pubs/pub51/
- MapReduce - https://research.google/pubs/pub62/
- Apache Hadoop Ecosystem - https://www.cloudera.com/products/open-source/apache-hadoop.html
- Apache Spark - https://spark.apache.org/
- Scala - https://www.scala-lang.org/

Bibliography

- Bernard Marr, "Data Strategy: How to Profit from a World of Big Data, Analytics and Artificial Intelligence", Kogan Page, 2nd edition, 2021.
- Florin Pop, Gabriel Neagu, "Big Data Platforms and Applications: Case Studies, Methods, Techniques, and Performance Evaluation (Computer Communications and Networks)", Springer; 1st edition, 2021.