Big Data Tools and Techniques

Chapter 7: Introduction to Spark (Part A)

Chapter Overview

Apache Spark is a lightning-fast cluster computing technology, designed for fast computation. Based on Hadoop MapReduce, it extends the MapReduce model to efficiently use it for more types of computations such as interactive queries and stream processing. The core feature of Spark is its in-memory cluster computing that increases the processing speed of an application.

Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming, thereby reducing the management burden of maintaining separate tools.

Chapter Overview (Objectives)

- Learn about Apache Spark's history and development
- Familiarize with Apache Spark as a lightning-fast cluster computing technology
- Appreciate Extract-Transform-Load operations, data analytics and visualization
- Understand the concepts: RDDs, DataFrames
- Know the implementation essentials - Transformations, Actions, pySpark, SparkSQL

Part 1: Spark Fundamentals

- Spark Background, Ops Mode, Components and Ecosystem
- Resilient Distributed Datasets (RDDs) – Fundamental Data Structure
- Write programs in terms of operations on distributed data
- Partitioned collections of objects spread across a cluster
- Diverse set of parallel transformations and actions
- Fault tolerance

Background (1/5)

Spark was introduced by Apache Software Foundation for speeding up the Hadoop computational software process.

Spark is not a modified version of Hadoop and is not dependent on Hadoop because it has its own cluster management. Hadoop is simply one of the ways to run Spark.

Spark uses Hadoop in two ways – one is storage and second is processing. Since Spark has its own cluster management system, it uses Hadoop for storage purpose only.

Spark is one of Hadoop's sub project developed in 2009 in UC Berkeley's AMPLab by Matei Zaharia. It was Open Sourced in 2010 under a BSD license and was donated to Apache software foundation in 2013. Spark has become a top level Apache project from 2014.

Background (2/5)

Spark is for:

Scalable, efficient *analysis* of *Big Data*

Where does big data come from?

It's all happening online – could record every:
» Click
» Ad impression
» Billing event
» Fast Forward, pause,... » Server request
» Transaction
» Network message
» Fault

User Generated Content (Web & Mobile)
» Facebook
» Instagram
»Yelp
» TripAdvisor » Twitter
» YouTube »...

Background (3/5)

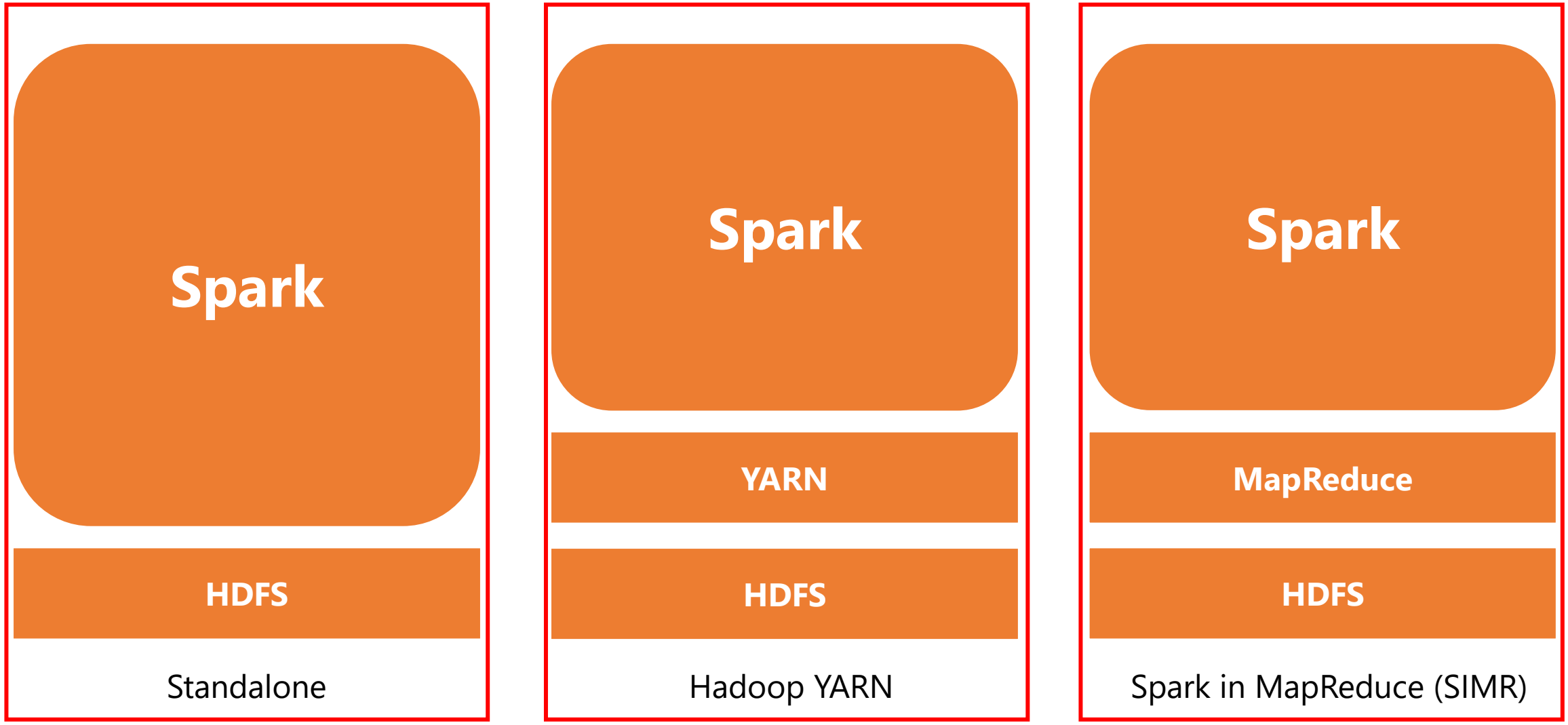
Features

Speed – Spark helps to run on top of the Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible through reducing number of read/write operations to disk as it stores the intermediate results in memory.

Supports multiple languages – Spark provides built-in APIs in Java, Scala, or Python and has approximately 80 high-level operators for interactive querying.

Advanced Analytics – Spark not only supports 'Map' and 'Reduce' operations. It also supports SQL queries, Streaming Data, Machine Learning (ML), and Graph Analytics.

Background (4/5)



Background (5/5)

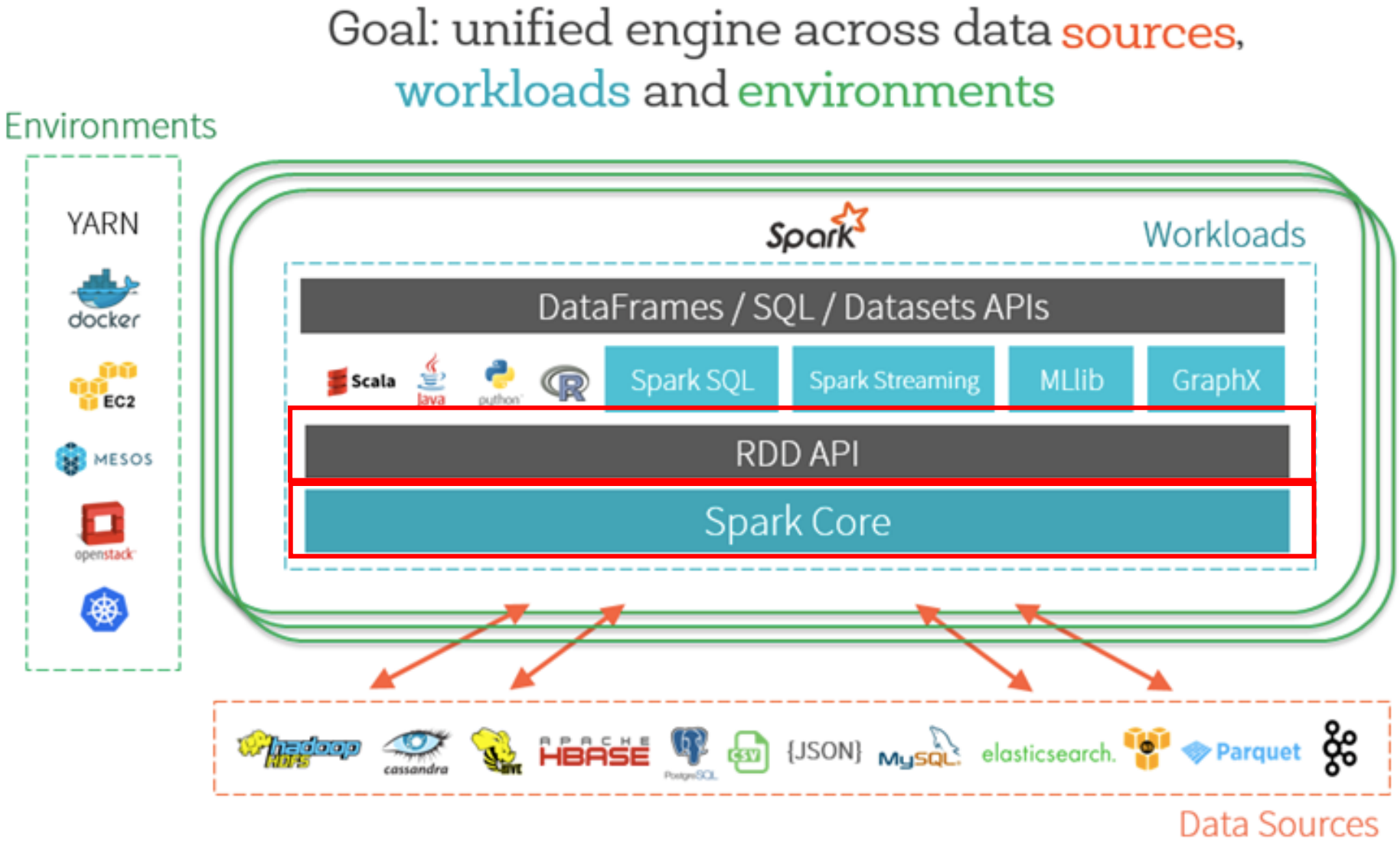
There are three ways of Spark deployment as follows.

Standalone – Spark Standalone deployment means Spark occupies the place on top of HDFS(Hadoop Distributed File System). In this case, Spark and MapReduce will run side by side to cover all spark jobs on cluster.

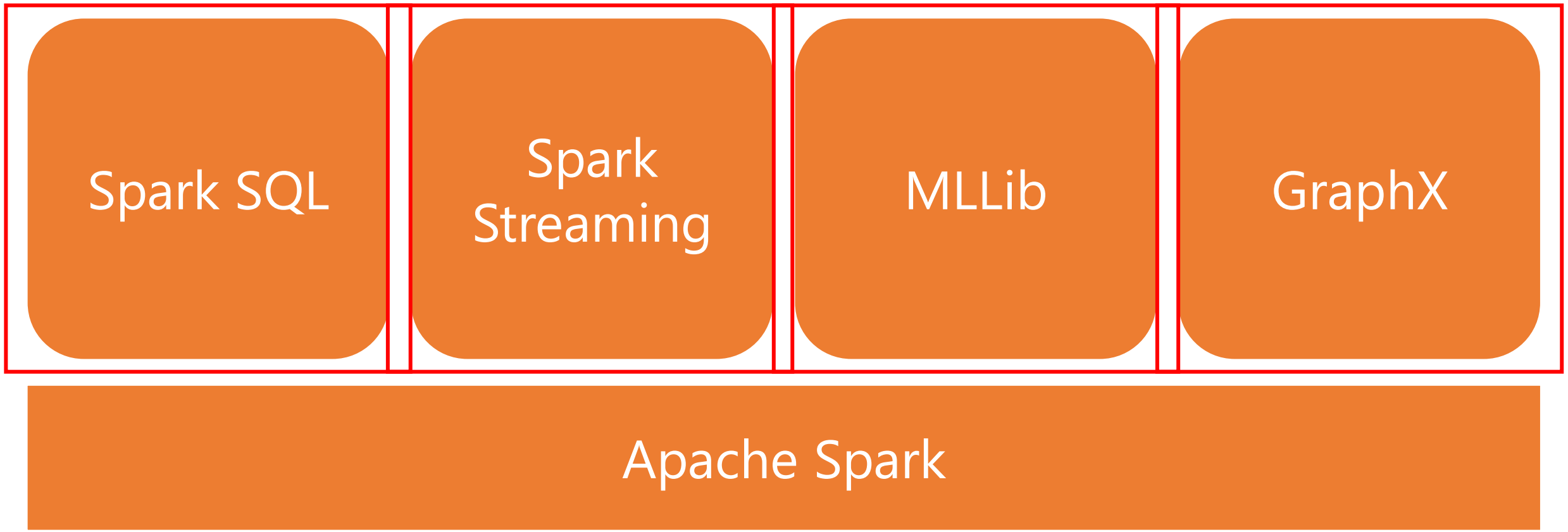
Hadoop Yarn – In this case, Spark simply runs on Yarn without any pre-installation or root access required. It helps to integrate Spark into Hadoop ecosystem or Hadoop stack. It allows other components to run on top of stack.

Spark in MapReduce (SIMR) – Spark in MapReduce is used to launch spark job in addition to standalone deployment.

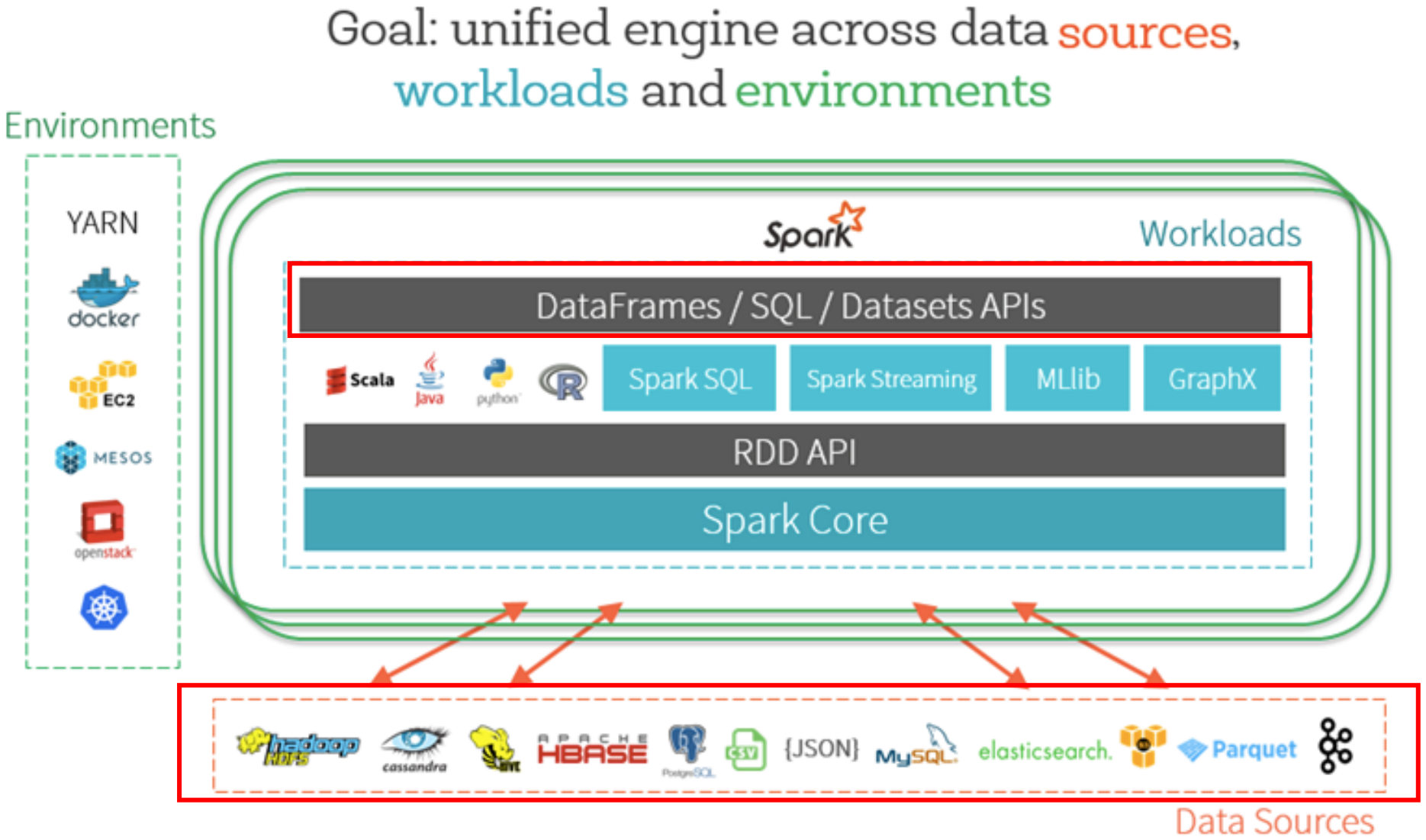
Spark Ecosystem



Spark Ecosystem



Spark Ecosystem



Spark Components (1/2)

Apache Spark Core

Spark Core is the underlying execution engine for Spark where all functionalities are built upon. It provides in-memory computing while referencing datasets in external file storage systems.

Spark SQL

Spark SQL is a component on top of Spark Core that comprises a new data abstraction called SchemaRDD, which provides support for structured and semi-structured data.

Spark Streaming

Spark Streaming uses Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in micro-batches and performs RDD (Resilient Distributed Datasets) transformations on those micro-batches of data.

Spark Components (2/2)

MLlib (Machine Learning Library)

MLlib is a distributed machine learning framework above Spark because of the distributed memory-based Spark architecture. Spark MLlib is nine times as fast as the Hadoop disk-based version of Apache Mahout.

GraphX

GraphX is a distributed graph-processing framework on top of Spark. It provides an API for representing graph computation that can model the user-defined graphs and provides an optimized runtime.

Resilient Distributed Dataset - RDD

A fundamental data structure of Spark. Immutable collection divided into logical partitions, which may be computed on different nodes of the cluster. Can contain any type of Python, Java, or Scala objects.

Read only, partitioned collection of records that is fault-tolerant and can be operated on in parallel.

Two ways to create RDDs − parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared file system.

RDD helps Spark achieve faster and efficient MapReduce operations.

Data Sharing is Slow in MapReduce

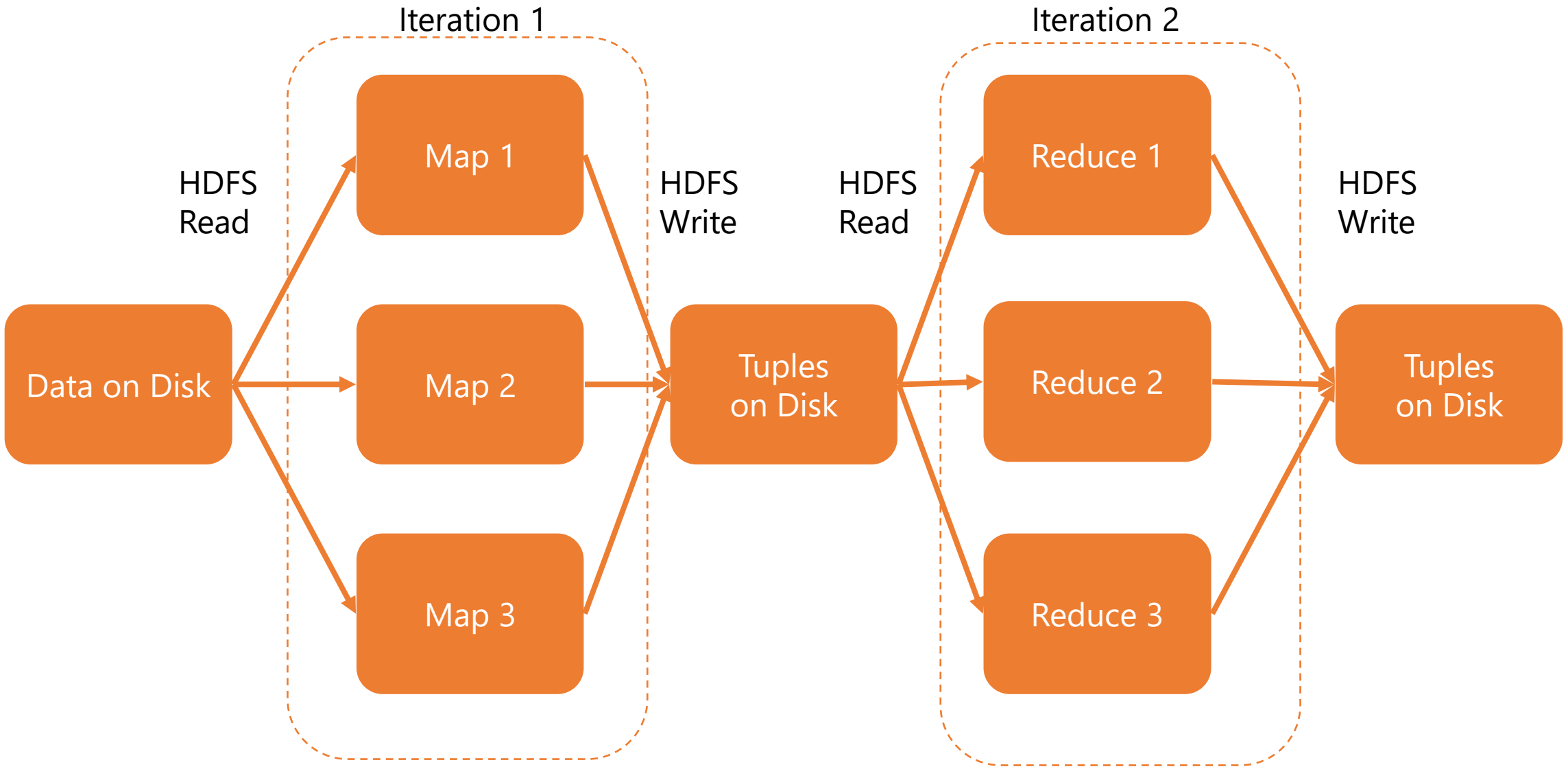
MapReduce is popularly used for processing and generating large datasets with a parallel, distributed algorithm on a cluster.

Unfortunately, the only way to reuse data between computations (between two MapReduce jobs) is to write it to an external stable storage system (e.g. HDFS).

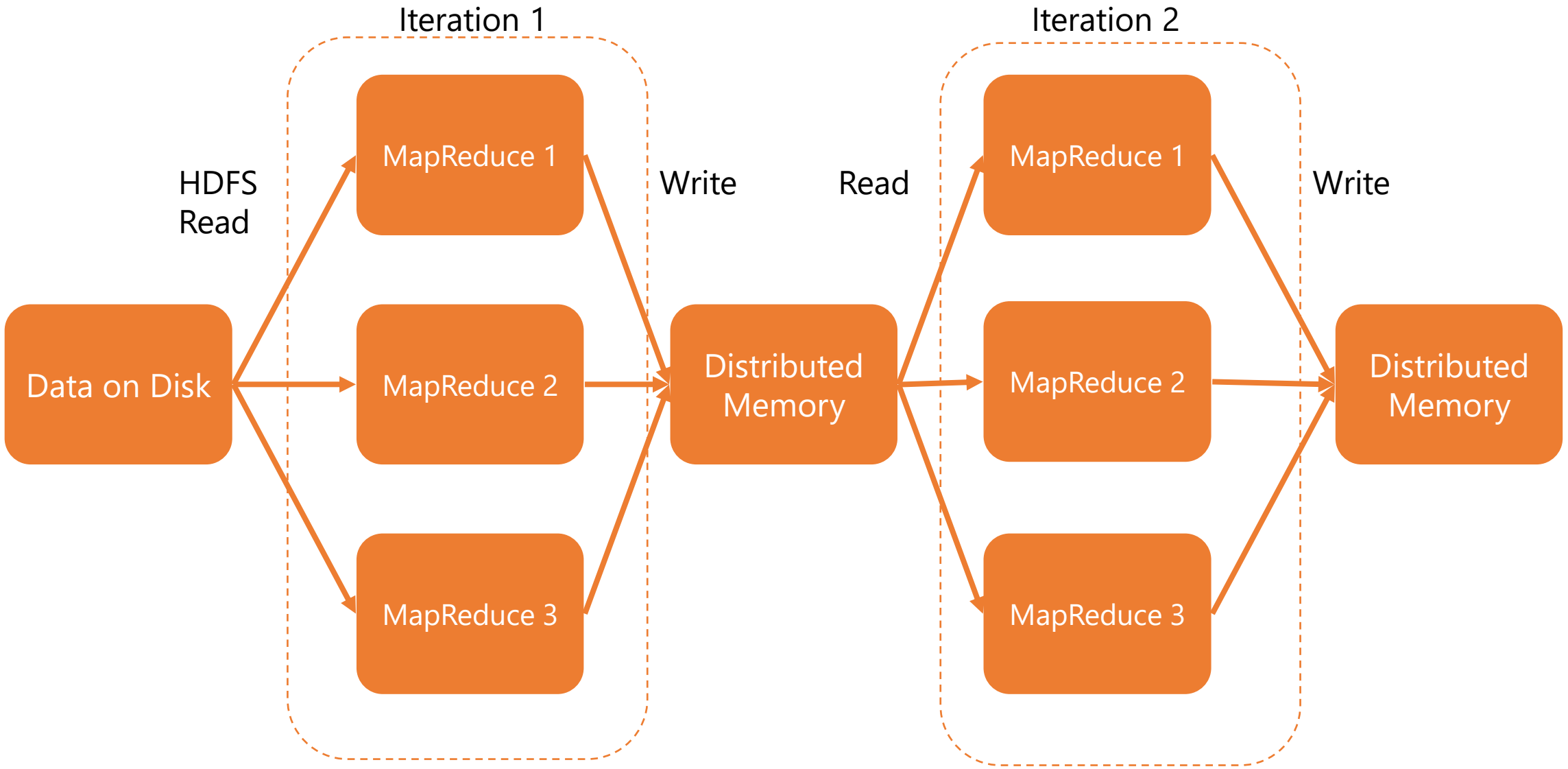
While iterative applications need more efficient data sharing across parallel jobs, MapReduce is slow due to replication, serialization, and disk IO.

As such, Hadoop applications spend more than 90% of the time performing HDFS read-write operations.

Iterative Ops on MapReduce



Iterative Ops on Spark RDD



Setting Up Spark (1/3)

1. **Verify Java Installation**

```
java -version
```

```
java version "1.7.0_71"
Java(TM) SE Runtime Environment (build 1.7.0_71-b13)
Java HotSpot(TM) Client VM (build 25.0-b02, mixed mode)
```

2. Verifying Scala Installation

```
scala -version
```

```
Scala code runner version 2.11.6 -- Copyright 2002-2013, LAMP/EPFL
```

Setting Up Spark (2/3)

3. Downloading Scala

Download the latest version of Scala software from the website https://www.scala-lang.org/download/.

4. Installing Scala

```
tar xvf scala-2.11.6.tgz
cd /home/Hadoop/Downloads/
mv scala-2.11.6 /usr/local/scala
exit
```

Use the following command for setting PATH for Scala.

```
export PATH = $PATH:/usr/local/scala/bin
```

```
scala -version
```

```
Scala code runner version 2.11.6 -- Copyright 2002-2013, LAMP/EPFL
```

Setting Up Spark (3/3)

5. Downloading Spark

Download the latest version of Apache Spark software from the website https://spark.apache.org/downloads.html

6. Installing Spark

```
tar xvf spark-1.3.1-bin-hadoop2.6.tgz
cd /home/Hadoop/Downloads/
mv spark-1.3.1-bin-hadoop2.6 /usr/local/spark
exit
```

Add the following line to ~/.bashrc file.

```
export PATH=$PATH:/usr/local/spark/bin
```

```
source ~/.bashrc
```

```
spark-shell
```

Spark Core

Spark Core provides

- distributed task dispatching, scheduling, and basic I/O functionalities.

- primary and fundamental abstraction known as Resilient Distributed Dataset (RDD)

- an interactive shell – a powerful tool to analyze data interactively. It is available in either Scala or Python language.

Spark Core - Shell

Opening the Spark Shell

The following command is used to open Spark shell.

```
$ spark-shell
```

Let us create a simple RDD from the text file.

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

The output for the above command is as follows.

```
inputfile: org.apache.spark.rdd.RDD[String] = input.txt MappedRDD[1] at
textFile at <console>:12
```

Spark Core – RDD Transformations

RDD transformations allows us to

- create dependencies between RDDs where each RDD in dependency has a function for calculating its data and has a pointer to its parent RDD.

- support lazy transformations where nothing will be implemented until an action is called to trigger job creation and execution.

Note that RDD transformation is not a set of data but is a step in a program telling Spark how to get data and what to do with it.

Examples of RDD transformations include:

`map(func), filter(func), flatMap(func), sample(withReplacement, fraction, seed), union(otherDataset), intersection(otherDataset), distinct, groupByKey, reduceByKey, aggregateByKey , sortByKey, join etc…`

Spark Core – RDD Transformations

RDD transformations allows us to

- create dependencies between RDDs where each RDD in dependency has a function for calculating its data and has a pointer to its parent RDD.

- support lazy transformations where nothing will be implemented until an action is called to trigger job creation and execution.

Note that RDD transformation is not a set of data but is a step in a program telling Spark how to get data and what to do with it.

Examples of RDD transformations include:

`map(func)`, `filter(func)`, `flatMap(func)`, `sample(withReplacement, fraction, seed)`, `union(otherDataset)`, `intersection(otherDataset)`, `distinct`, `groupByKey`, `reduceByKey`, `aggregateByKey` , `sortByKey`, `join` etc…

Spark Core – RDD Transformations

RDD transformations allows us to

- create dependencies between RDDs where each RDD in dependency has a function for calculating its data and has a pointer to its parent RDD.

- support lazy transformations where nothing will be implemented until an action is called to trigger job creation and execution.

Note that RDD transformation is not a set of data but is a step in a program telling Spark how to get data and what to do with it.

Examples of RDD transformations include:

`map(func), filter(func), flatMap(func), sample(withReplacement, fraction, seed), union(otherDataset), intersection(otherDataset), distinct, groupByKey, reduceByKey, aggregateByKey, sortByKey, join` etc…

Spark Core – RDD Actions

Actions trigger job creation and execution of the transformations.

Given here a list of Actions, which return results.

Examples of RDD actions include:

`reduce(func)`, `collect()`, `count()`, `first()`, `take(n)`, `takeSample (withReplacement,num, [seed])`, `takeOrdered(n, [ordering])`, `saveAsTextFile(path)`, `countByKey()`, `foreach(func)` etc…

Spark Core – RDD Actions

Actions trigger job creation and execution of the transformations.

Given here a list of Actions, which return results.

Examples of RDD actions include:

`reduce(func), collect(), count(), first(), take(n), takeSample (withReplacement,num, [seed]), takeOrdered(n, [ordering]), saveAsTextFile(path), countByKey(), foreach(func)` etc…

Spark Core Programming (1/5)

Let us walk through an example on transformations and actions in RDD programming.

Consider a word count application which counts each word appearing in a file. Let the following text be an input which is to be stored on hdfs.

`input.txt – input file.`

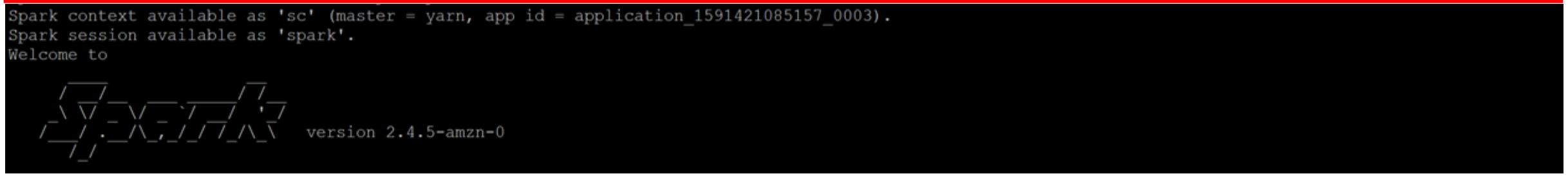
```
Learning how to ride a bicycle for the first time was a nerve racking
independent moment. I was about eight years old when my sister informed me that
I was too old to still be riding a bicycle with training wheels.
```

Follow the next steps to execute the given example.

Spark Core Programming (2/5)

The following command is used to open Spark shell.

```
$ spark-shell
```

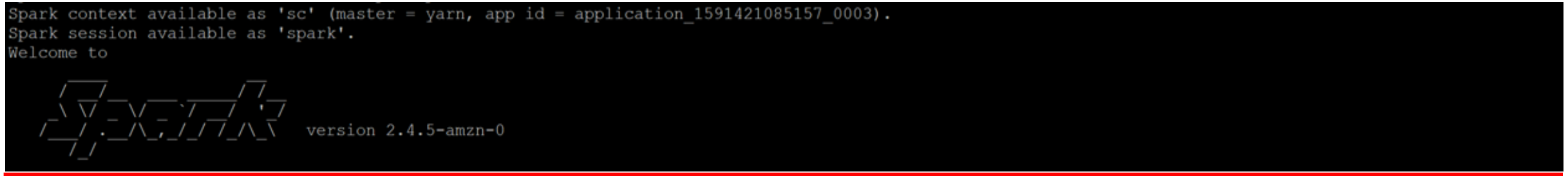



```
scala> val inputfile = sc.textFile("input.txt")

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

scala> counts.toDebugString

scala> counts.cache()

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

Spark Core Programming (2/5)

The following command is used to open Spark shell.

```
$ spark-shell
```

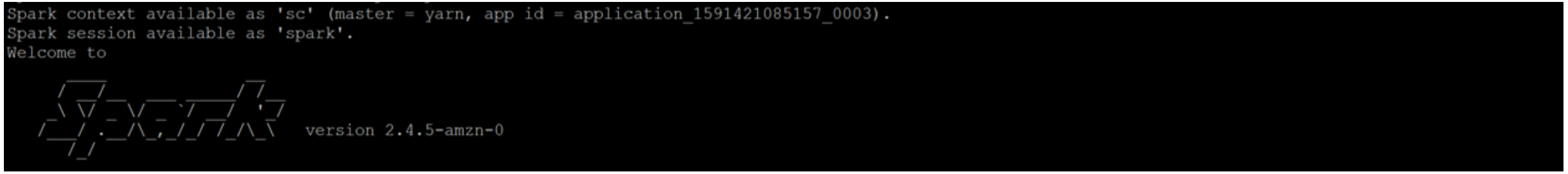



```
scala> val inputfile = sc.textFile("input.txt")

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

scala> counts.toDebugString

scala> counts.cache()

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

Spark Core Programming (2/5)

The following command is used to open Spark shell.

```
$ spark-shell
```

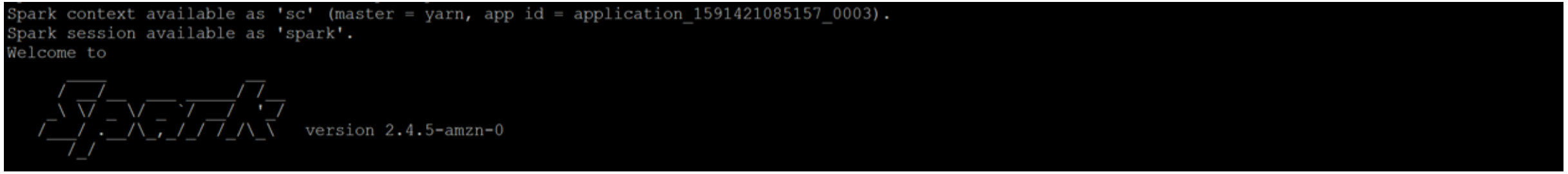



```
scala> val inputfile = sc.textFile("input.txt")

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

scala> counts.toDebugString

scala> counts.cache()

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

Spark Core Programming (2/5)

The following command is used to open Spark shell.

```
$ spark-shell
```

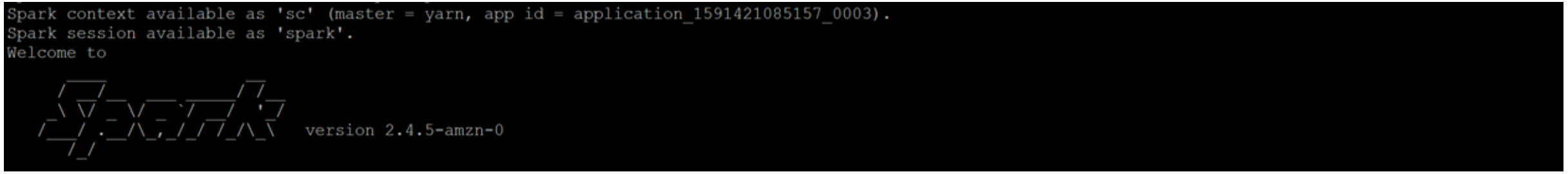



```
scala> val inputfile = sc.textFile("input.txt")

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

scala> counts.toDebugString

scala> counts.cache()

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

Spark Core Programming (2/5)

The following command is used to open Spark shell.

```
$ spark-shell
```

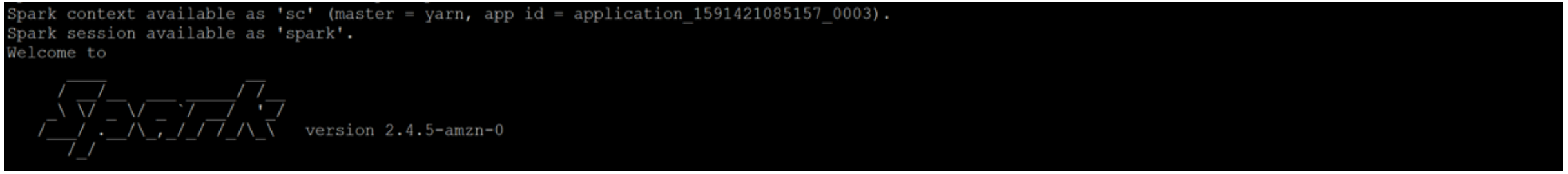



```
scala> val inputfile = sc.textFile("input.txt")

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

scala> counts.toDebugString

scala> counts.cache()

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

Spark Core Programming (2/5)

The following command is used to open Spark shell.

```
$ spark-shell
```




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

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

scala> counts.toDebugString

scala> counts.cache()

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

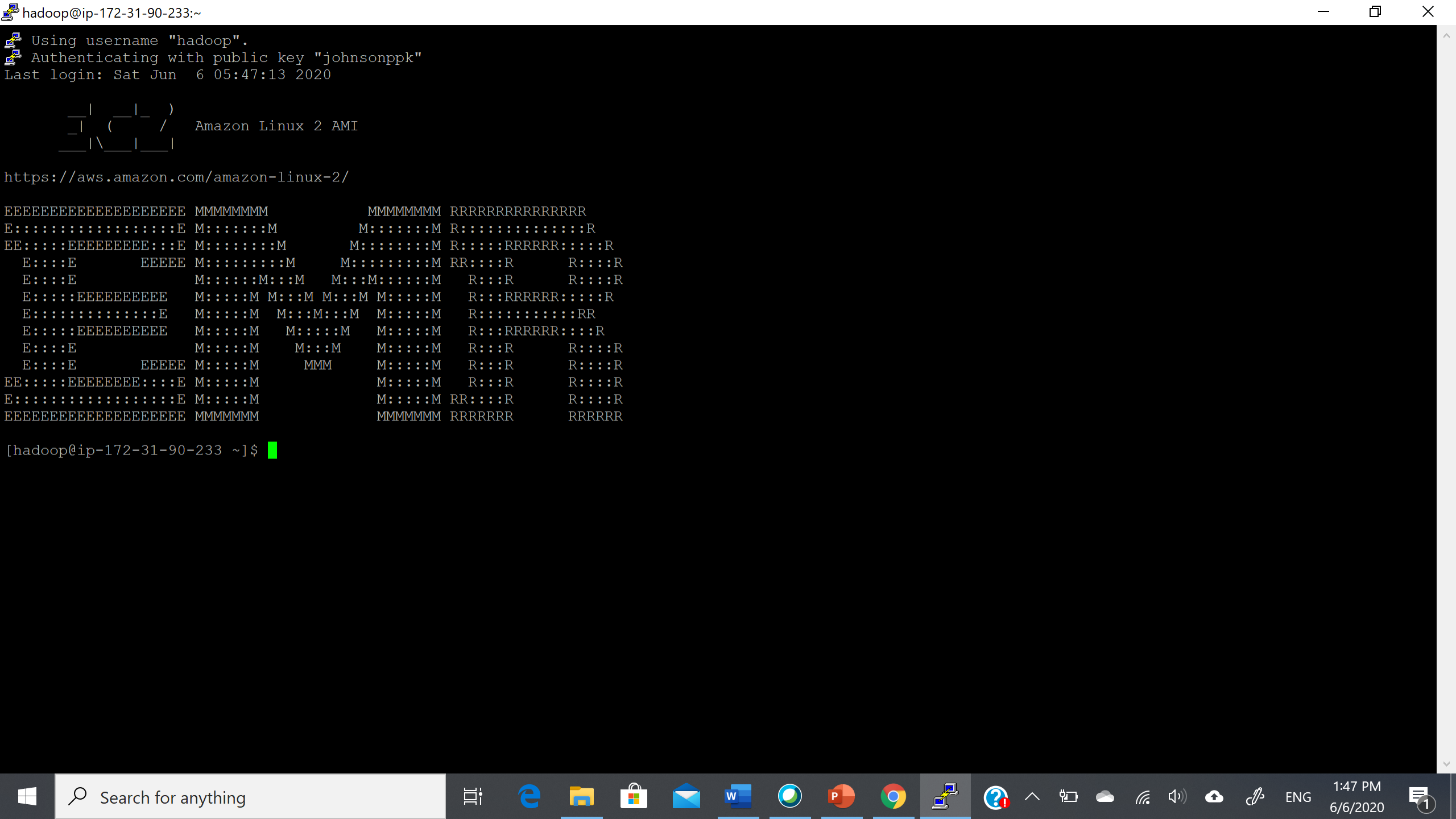
Spark Core Programming (3/5)

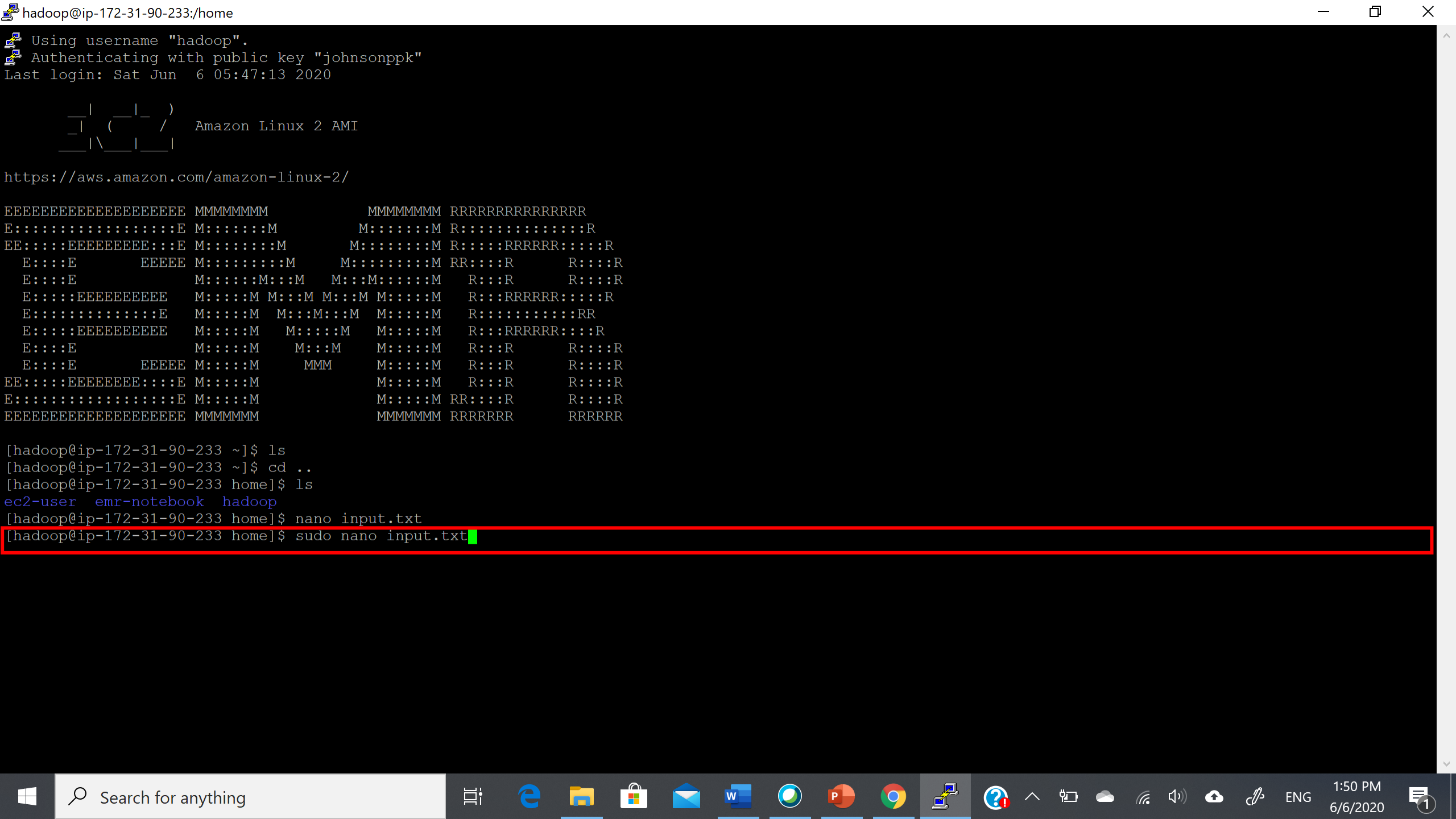
```
#Checking the Output
```

```
[hadoop@localhost ~]$ hadoop fs -ls output
```

```
part-00000
part-00001
_SUCCESS
```

```
[hadoop@localhost output]$ hadoop fs -cat output/part-00000
[hadoop@localhost output]$ hadoop fs -cat output/part-00001
```


hadoop@ip-172-31-90-233:~
Using username "hadoop".
Authenticating with public key "johnsonppk"
Last login: Sat Jun 6 05:47:13 2020
__| __|_)
_| (/ Amazon Linux 2 AMI
___|___|___|
https://aws.amazon.com/amazon-linux-2/
EEEEEEEEEEEEEEEEEEE MMMMMMMM MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M::::::M M::::::M R::::::::::::::R
EE::::EEEEEEEEE:::E M:::::::M M:::::::M R::::RRRRRR:::::R
E::::E EEEEE M::::::::M M::::::::M RR::::R R::::R
E::::E M::::::M:::M M:::M::::::M R:::R R::::R
E::::::EEEEEEEEEE M:::::M M:::M M:::M M:::::M R:::RRRRRR:::::R
E::::::::::::::E M:::::M M:::M:::M M:::::M R::::::::::RR
E::::::EEEEEEEEEE M:::::M M:::::M M:::::M R:::RRRRRR::::R
E::::E M:::::M M:::M M:::::M R:::R R::::R
E::::E EEEEE M:::::M MMM M:::::M R:::R R::::R
EE::::EEEEEEEE::::E M:::::M M:::::M R:::R R::::R
E::::::::::::::::::E M:::::M M:::::M RR::::R R::::R
EEEEEEEEEEEEEEEEEEE MMMMMM MMMMMM RRRRRRR RRRRRR
[hadoop@ip-172-31-90-233 ~]$
Search for anything
ENG
1:47 PM
6/6/2020


hadoop@ip-172-31-90-233:/home
Using username "hadoop".
Authenticating with public key "johnsonppk"
Last login: Sat Jun 6 05:47:13 2020
Amazon Linux 2 AMI
https://aws.amazon.com/amazon-linux-2/
[hadoop@ip-172-31-90-233 ~]$ ls
[hadoop@ip-172-31-90-233 ~]$ cd ..
[hadoop@ip-172-31-90-233 home]$ ls
ec2-user emr-notebook hadoop
[hadoop@ip-172-31-90-233 home]$ nano input.txt
[hadoop@ip-172-31-90-233 home]$ sudo nano input.txt
Search for anything
ENG
1:50 PM
6/6/2020

hadoop@ip-172-31-90-233:/home

GNU nano 2.9.8 input.txt

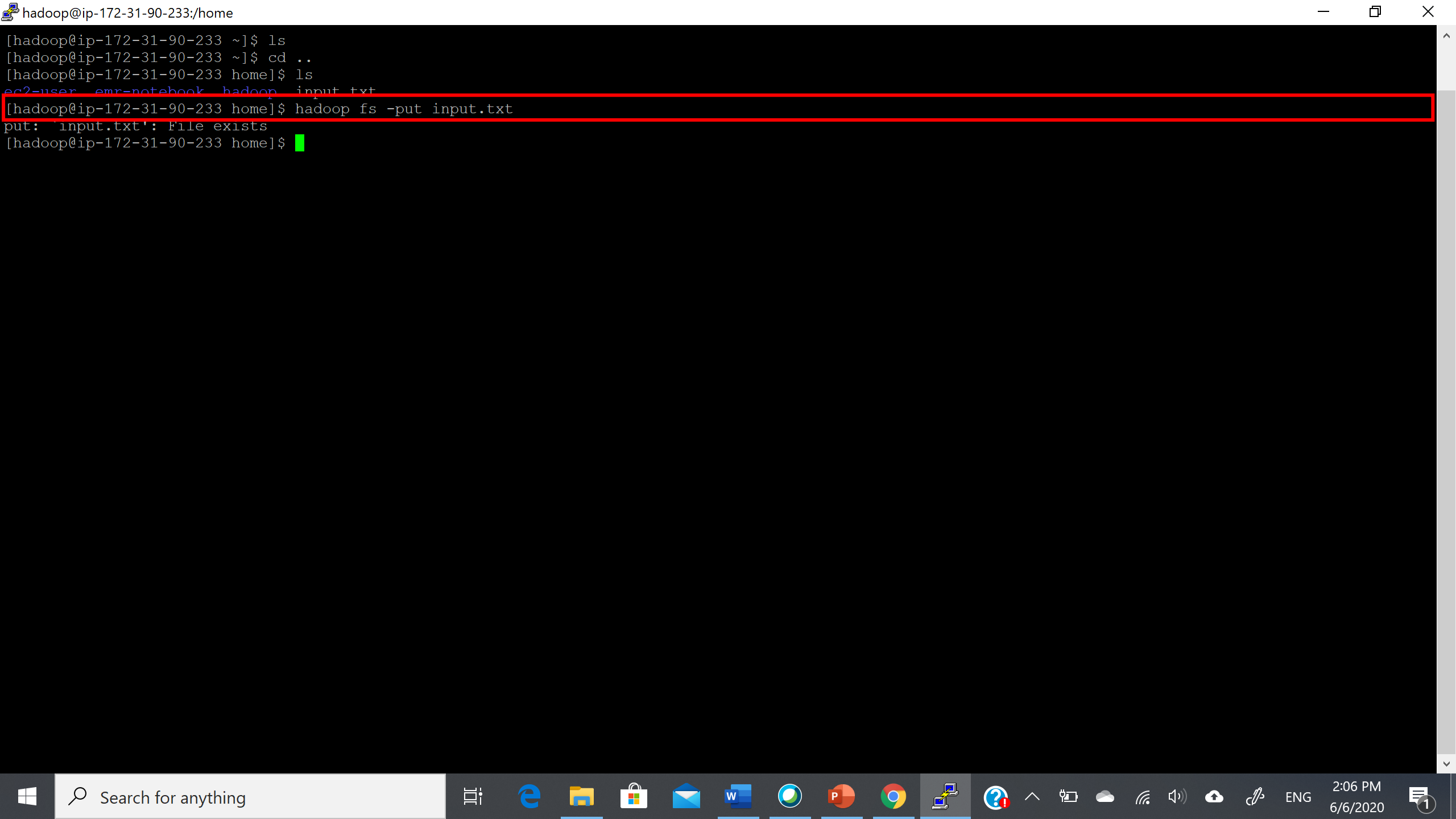
Learning how to ride a bicycle for the first time was a nerve racking independent moment. I was about eight years old when my sister informed me that I was $

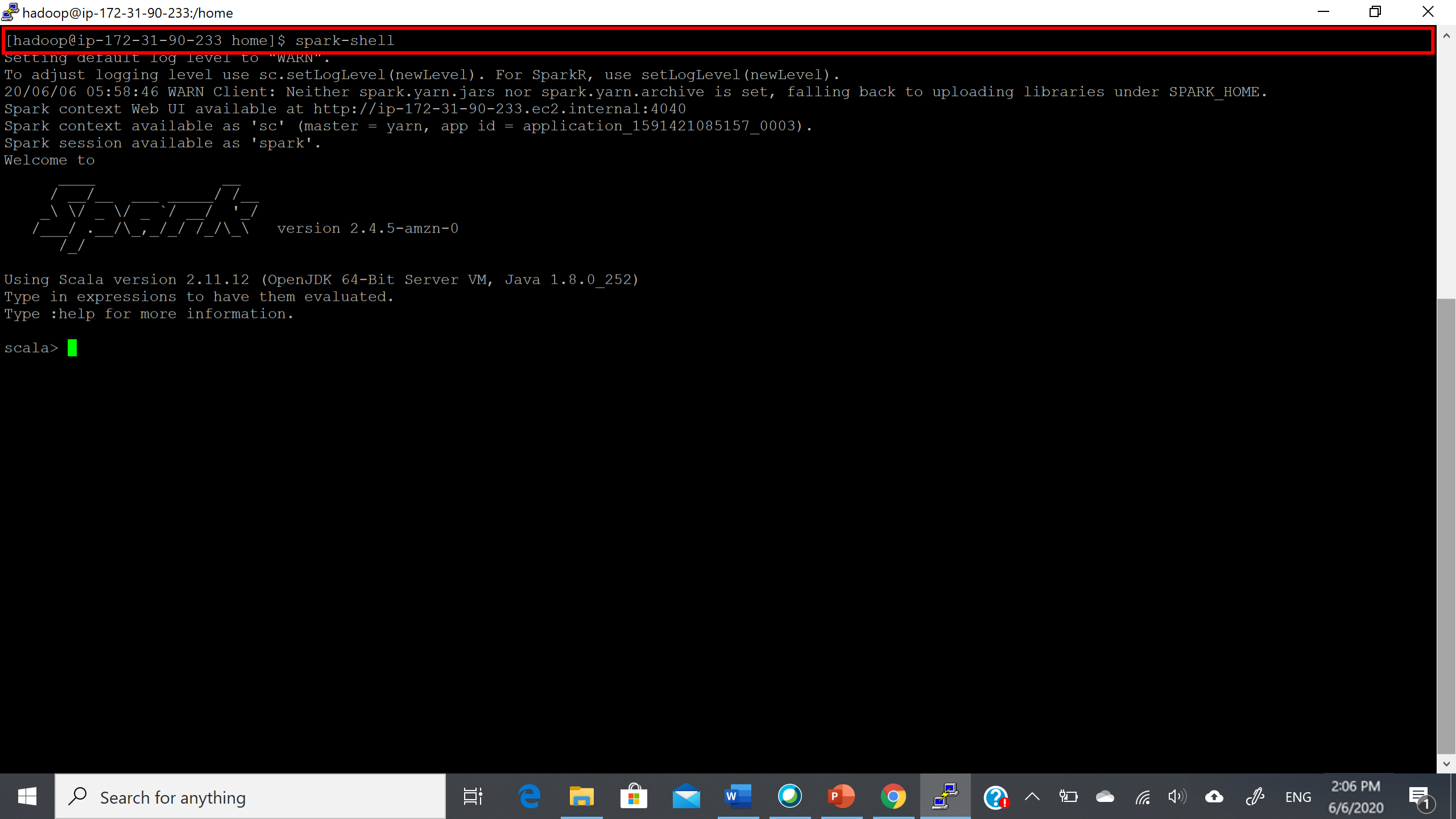
[Read 2 lines]

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos M-U Undo M-A Mark Text M-] To Bracket
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line M-E Redo M-6 Copy Text M-W WhereIs Next

Search for anything

ENG 1:51 PM 6/6/2020


hadoop@ip-172-31-90-233:/home
[hadoop@ip-172-31-90-233 ~]$ ls
[hadoop@ip-172-31-90-233 ~]$ cd ..
[hadoop@ip-172-31-90-233 home]$ ls
ec2-user emr-notebook hadoop input.txt
[hadoop@ip-172-31-90-233 home]$ hadoop fs -put input.txt
put: `input.txt': File exists
[hadoop@ip-172-31-90-233 home]$
Search for anything
ENG
2:06 PM
6/6/2020


hadoop@ip-172-31-90-233:/home
[hadoop@ip-172-31-90-233 home]$ spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
20/06/06 05:58:46 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.
Spark context Web UI available at http://ip-172-31-90-233.ec2.internal:4040
Spark context available as 'sc' (master = yarn, app id = application_1591421085157_0003).
Spark session available as 'spark'.
Welcome to
version 2.4.5-amzn-0
Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_252)
Type in expressions to have them evaluated.
Type :help for more information.
scala>
Search for anything
ENG
2:06 PM
6/6/2020

hadoop@ip-172-31-90-233:/home

```
[hadoop@ip-172-31-90-233 ~]$ ls
[hadoop@ip-172-31-90-233 ~]$ cd ..
[hadoop@ip-172-31-90-233 home]$ ls
ec2-user  emr-notebook  hadoop  input.txt
[hadoop@ip-172-31-90-233 home]$ hadoop fs -put input.txt
put: `input.txt': File exists
[hadoop@ip-172-31-90-233 home]$ spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
20/06/06 06:07:05 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.
Spark context Web UI available at http://ip-172-31-90-233.ec2.internal:4040
Spark context available as 'sc' (master = yarn, app id = application_1591421085157_0004).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.5-amzn-0
      /_/

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_252)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val inputfile = sc.textFile("input.txt")
inputfile: org.apache.spark.rdd.RDD[String] = input.txt MapPartitionsRDD[1] at textFile at <console>:24

scala> val counts = inputfile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(_+_);
counts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <console>:25

scala>
```

hadoop@ip-172-31-90-233:/home

```
[hadoop@ip-172-31-90-233 ~]$ ls
[hadoop@ip-172-31-90-233 ~]$ cd ..
[hadoop@ip-172-31-90-233 home]$ ls
ec2-user  emr-notebook  hadoop  input.txt
[hadoop@ip-172-31-90-233 home]$ hadoop fs -put input.txt
put: `input.txt': File exists
[hadoop@ip-172-31-90-233 home]$ spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
20/06/06 06:07:05 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.
Spark context Web UI available at http://ip-172-31-90-233.ec2.internal:4040
Spark context available as 'sc' (master = yarn, app id = application_1591421085157_0004).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.5-amzn-0
      /_/

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_252)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val inputfile = sc.textFile("input.txt")
inputfile: org.apache.spark.rdd.RDD[String] = input.txt MapPartitionsRDD[1] at textFile at <console>:24

scala> val counts = inputfile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(_+_);
counts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <console>:25

scala> counts.toDebugString
res0: String =
(2) ShuffledRDD[4] at reduceByKey at <console>:25 []
 +-(2) MapPartitionsRDD[3] at map at <console>:25 []
    |  MapPartitionsRDD[2] at flatMap at <console>:25 []
    |  input.txt MapPartitionsRDD[1] at textFile at <console>:24 []
    |  input.txt HadoopRDD[0] at textFile at <console>:24 []

scala> counts.cache()
res1: counts.type = ShuffledRDD[4] at reduceByKey at <console>:25

scala>
```

hadoop@ip-172-31-90-233:/home

```
[hadoop@ip-172-31-90-233 ~]$ cd ..
[hadoop@ip-172-31-90-233 home]$ ls
ec2-user  emr-notebook  hadoop  input.txt
[hadoop@ip-172-31-90-233 home]$ hadoop fs -put input.txt
put: `input.txt': File exists
[hadoop@ip-172-31-90-233 home]$ spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
20/06/06 06:07:05 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.
Spark context Web UI available at http://ip-172-31-90-233.ec2.internal:4040
Spark context available as 'sc' (master = yarn, app id = application_1591421085157_0004).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.5-amzn-0
      /_/

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_252)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val inputfile = sc.textFile("input.txt")
inputfile: org.apache.spark.rdd.RDD[String] = input.txt MapPartitionsRDD[1] at textFile at <console>:24

scala> val counts = inputfile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(_+_);
counts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <console>:25

scala> counts.toDebugString
res0: String =
(2) ShuffledRDD[4] at reduceByKey at <console>:25 []
 +-(2) MapPartitionsRDD[3] at map at <console>:25 []
    |  MapPartitionsRDD[2] at flatMap at <console>:25 []
    |  input.txt MapPartitionsRDD[1] at textFile at <console>:24 []
    |  input.txt HadoopRDD[0] at textFile at <console>:24 []

scala> counts.cache()
res1: counts.type = ShuffledRDD[4] at reduceByKey at <console>:25

scala> counts.saveAsTextFile("output")

scala>
```

hadoop@ip-172-31-90-233:/home

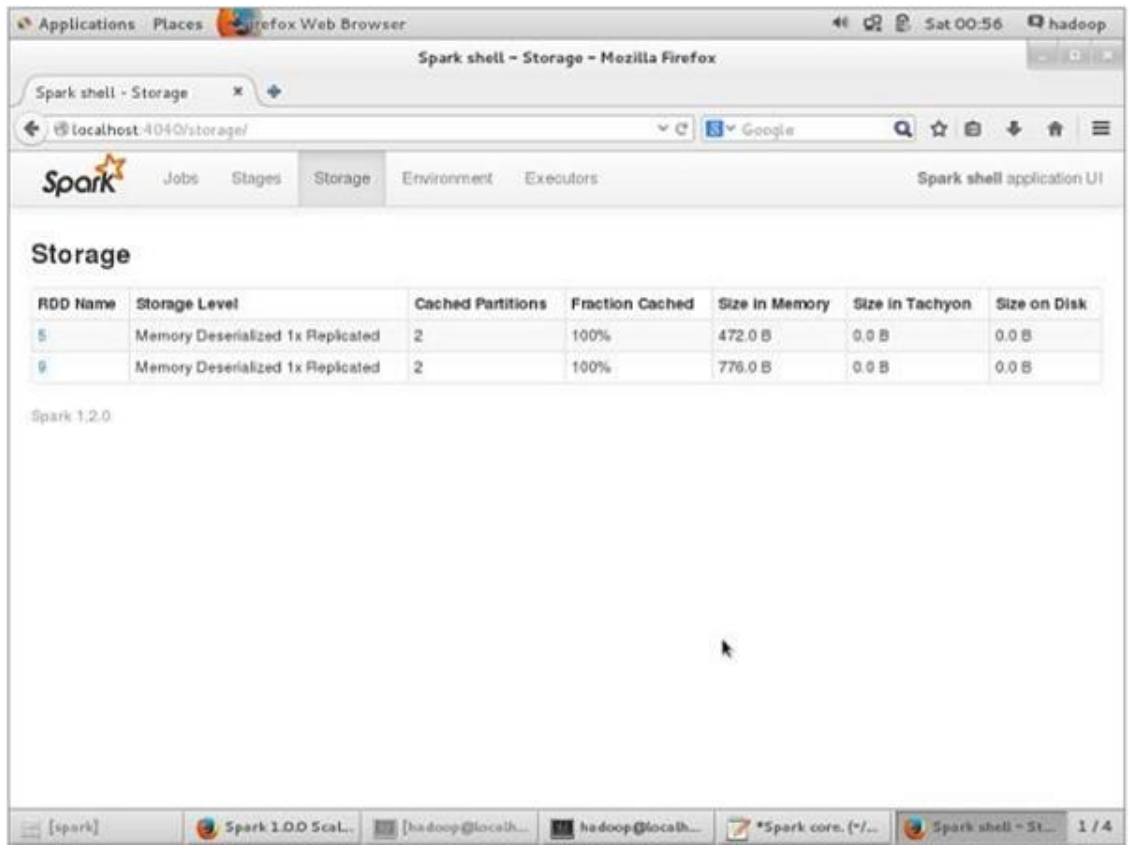
```
[hadoop@ip-172-31-90-233 home]$ hadoop fs -ls output
Found 3 items
-rw-r--r--   1 hadoop hadoop          0 2020-06-06 06:10 output/_SUCCESS
-rw-r--r--   1 hadoop hadoop        192 2020-06-06 06:10 output/part-00000
-rw-r--r--   1 hadoop hadoop        131 2020-06-06 06:10 output/part-00001
[hadoop@ip-172-31-90-233 home]$
```

hadoop@ip-172-31-90-233:/home

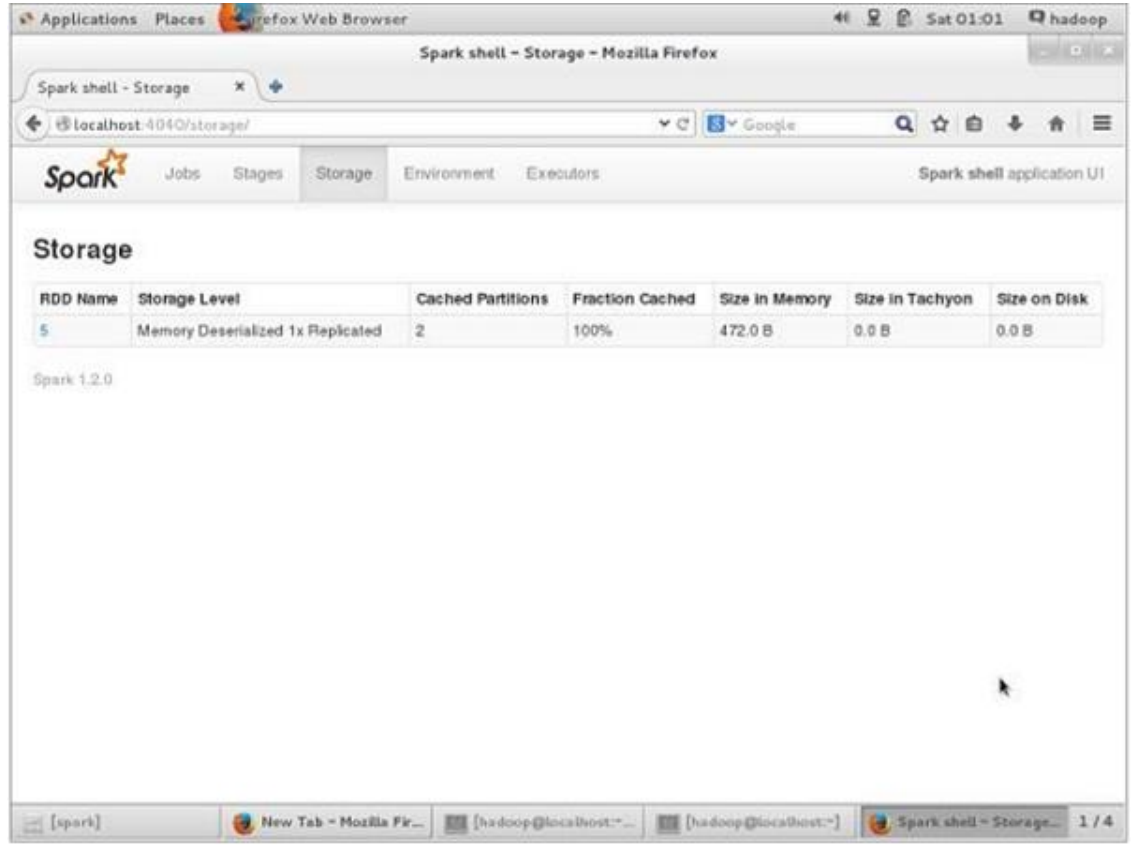
```
[hadoop@ip-172-31-90-233 home]$ hadoop fs -cat output/part-00001
(about,1)
(was,3)
(that,1)
(bicycle,2)
(a,3)
(old,2)
(be,1)
(eight,1)
(riding,1)
(I,2)
(to,2)
(racking,1)
(for,1)
(time,1)
(the,1)
[hadoop@ip-172-31-90-233 home]$
```

Spark Core Programming (4/5)

`http://<ip-address>:4040`



`Scala> counts.unpersist()`



Spark Core Setting Programming Up PySpark (5/5)

1. Installing PySpark

Before installing PySpark, we must have Python and Spark installed.

```
wget https://repo.anaconda.com/archive/Anaconda3-5.1.0-Linux-x86_64.sh
pip install — upgrade pip
pip install pyspark
pyspark
```

hadoop@ip-172-31-90-233:/home

```
[hadoop@ip-172-31-90-233 home]$ pyspark
Python 3.7.6 (default, Feb 26 2020, 20:54:15)
[GCC 7.3.1 20180712 (Red Hat 7.3.1-6)] on linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
20/06/06 06:32:34 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.5-amzn-0
      /_/

Using Python version 3.7.6 (default, Feb 26 2020 20:54:15)
SparkSession available as 'spark'.
>>>
```