**Ruth Maina; DSC 650 Week4 Exercise:** Introduction to Apache Spark using Scala and PySpark

### Objective: Dive into Apache Spark’s capabilities using both PySpark and Scala in the respective shells, experimenting with data generation and transformations.

#### **1. Environment Initialization**

* As you did in the Week 2 and Week 3 assignments, begin by navigating to the appropriate directory:
* cd bellevue-bigdata  
  cd hadoop-hive-spark-hbase
* Start your Docker containers:
* docker-compose up -d
* Access the master container:
* docker-compose exec master bash

#### **2. Running a Built-in Spark Example with PySpark**

To get hands-on experience with PySpark, we will execute the provided SparkPi example using the following command in the master container:

spark-submit --class org.apache.spark.examples.SparkPi \  
 --master yarn \  
 --deploy-mode client \  
 --driver-memory 2g \  
 --executor-memory 1g \  
 --executor-cores 1 \  
 $SPARK\_HOME/examples/jars/spark-examples\*.jar \  
 10

**Deliverable:** Screenshot of the SparkPi output.

A screenshot of a computer screen

Description automatically generated

#### **3. Starting the Spark Scala Shell**

In the master container’s terminal, initiate the Spark Scala shell:

$SPARK\_HOME/bin/spark-shell --master yarn --driver-memory 2g --executor-memory 1g --executor-cores 1

#### **4. Generating and Printing Random Numbers in the Scala Shell**

Execute the following commands in the Spark Scala shell:

val numNumbers = 10000  
val numbers = (1 to numNumbers).map(\_ => scala.util.Random.nextInt(1000))  
val numbersRDD = sc.parallelize(numbers)  
numbersRDD.take(100).foreach(println)

**Deliverable:** Screenshot of the first 100 generated random numbers.

A screenshot of a computer

Description automatically generated

A black screen with a black border

Description automatically generated

#### **5. Generating and Transforming Random Sentences in the Scala Shell**

Generate random sentences and apply a transformation of your choice:

val numberOfSentences = 1000  
val words = List("apple", "banana", "cherry", "date", "elderberry", "fig", "grape", "honeydew")  
val sentences = (1 to numberOfSentences).map(\_ => scala.util.Random.shuffle(words).take(scala.util.Random.nextInt(6) + 1).mkString(" ") + ".")  
val sentencesRDD = sc.parallelize(sentences)  
  
// Apply your custom transformation here  
val transformedSentences = sentencesRDD.map(sentence => /\* Your transformation code here \*/)  
transformedSentences.take(100).foreach(println)

**Instructions:** Modify the placeholder /\* Your transformation code here \*/ to apply a unique transformation to each sentence.

**Deliverable:** Screenshot of a segment of the generated transformed sentences and an explanation of your unique transformation.

BEFORE TRANSFORMATION:

A screenshot of a computer

Description automatically generated

TRANSFORMATION STEPS EXPLANATION:

* ‘transformedSentences’ variable is created to hold the assignment value from the transformation
* take every sentence contained in the sentence list via the ‘sentence’ keyword
* Use .toUppercase after the sentence keyword to convert all sentences to uppercase
* Assign the uppercase sentences output to ‘transformedSentences’ variable
* Print the transformed sentences by referencing the ‘transformedSentences’ variable created earlier

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Week 4 Exercise feedback

You successfully ran the Spark submit command to calculate pi using Apache Spark. The command provided specific instructions for running the SparkPi example. Key arguments included:

--class: Specifies the main class for the Spark application (org.apache.spark.examples.SparkPi).

--master: Defines the cluster manager (set to YARN).

--deploy-mode: Determines the deployment mode (client mode for running the driver on the client machine).

--driver-memory: Allocates 2 gigabytes of memory for the driver process.

--executor-memory: Allocates 1 gigabyte of memory for the executor processes.

--executor-cores: Sets the number of CPU cores per executor to 1.

The command also included the path to the Spark example jar file and an argument 10, instructing the SparkPi application to calculate Pi with a precision of 10 decimal places.

Additionally, you successfully explored Spark Scala with number generation, working with Scala for tasks related to generating and processing numeric data.

Furthermore, you successfully explored custom word/sentence transformations, demonstrating your ability to manipulate text data within the Spark environment. Custom transformations can involve tasks like text preprocessing, sentiment analysis, or other specialized operations on words and sentences to meet specific analytical requirements.

Lastly, it's worth noting that Spark is invaluable for processing big data. It offers a distributed computing framework that enables efficient parallel processing and analysis of large datasets. Beyond its scalability, Spark provides a wide range of built-in libraries and APIs for various data processing tasks, making it a versatile tool for handling big data challenges effectively.