

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:

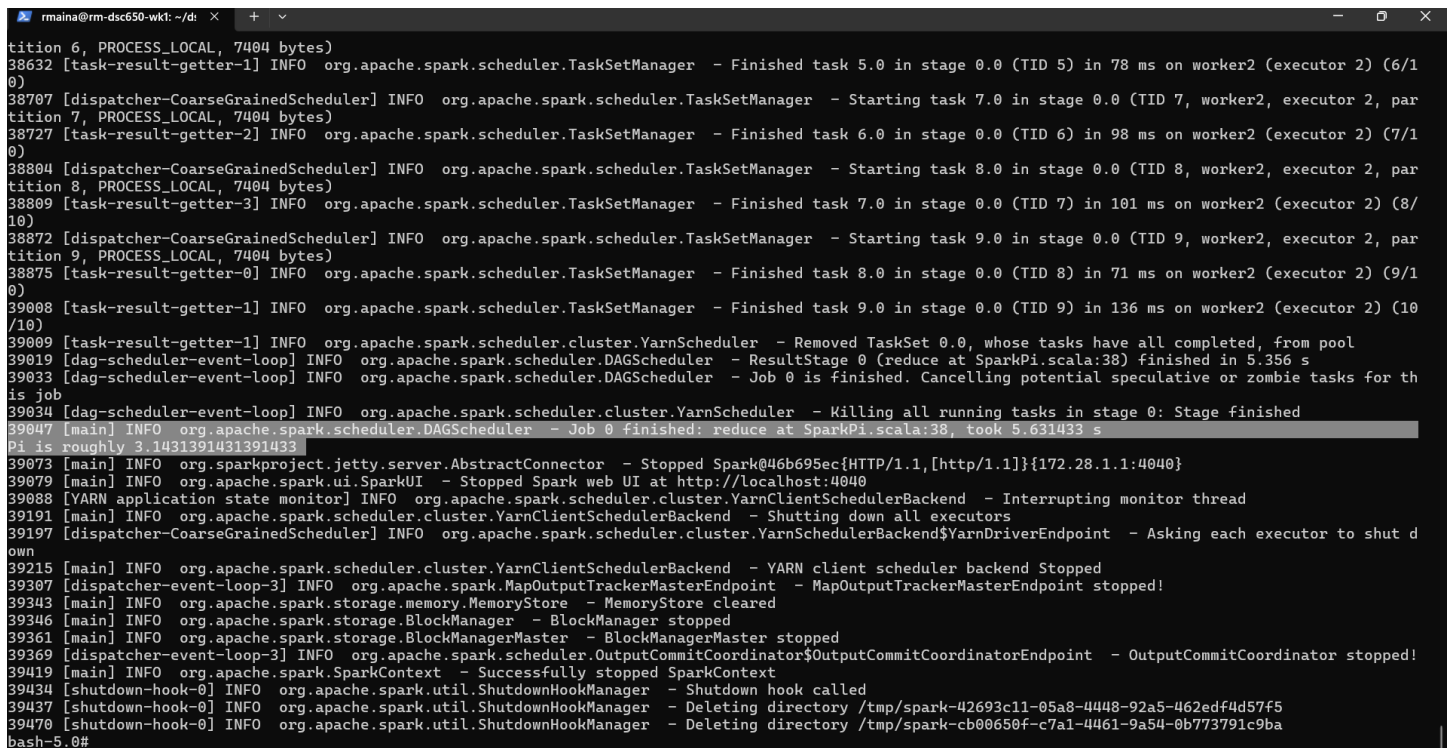
```
docker-compose exec master bash
```

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.



The screenshot shows a terminal window with the SparkPi output. The output displays the progress of the SparkPi job, including task completion times and resource usage. Key lines include:

```
38632 [task-result-getter-1] INFO org.apache.spark.scheduler.TaskSetManager - Finished task 5.0 in stage 0.0 (TID 5) in 78 ms on worker2 (executor 2) (6/10)
38707 [dispatcher-CoarseGrainedScheduler] INFO org.apache.spark.scheduler.TaskSetManager - Starting task 7.0 in stage 0.0 (TID 7, worker2, executor 2, partition 7, PROCESS_LOCAL, 7404 bytes)
38727 [task-result-getter-2] INFO org.apache.spark.scheduler.TaskSetManager - Finished task 6.0 in stage 0.0 (TID 6) in 98 ms on worker2 (executor 2) (7/10)
38804 [dispatcher-CoarseGrainedScheduler] INFO org.apache.spark.scheduler.TaskSetManager - Starting task 8.0 in stage 0.0 (TID 8, worker2, executor 2, partition 8, PROCESS_LOCAL, 7404 bytes)
38809 [task-result-getter-3] INFO org.apache.spark.scheduler.TaskSetManager - Finished task 7.0 in stage 0.0 (TID 7) in 101 ms on worker2 (executor 2) (8/10)
38872 [dispatcher-CoarseGrainedScheduler] INFO org.apache.spark.scheduler.TaskSetManager - Starting task 9.0 in stage 0.0 (TID 9, worker2, executor 2, partition 9, PROCESS_LOCAL, 7404 bytes)
38875 [task-result-getter-0] INFO org.apache.spark.scheduler.TaskSetManager - Finished task 8.0 in stage 0.0 (TID 8) in 71 ms on worker2 (executor 2) (9/10)
39008 [task-result-getter-1] INFO org.apache.spark.scheduler.TaskSetManager - Finished task 9.0 in stage 0.0 (TID 9) in 136 ms on worker2 (executor 2) (10/10)
39009 [task-result-getter-1] INFO org.apache.spark.scheduler.cluster.YarnScheduler - Removed TaskSet 0.0, whose tasks have all completed, from pool
39019 [dag-scheduler-event-loop] INFO org.apache.spark.scheduler.DAGScheduler - ResultStage 0 (reduce at SparkPi.scala:38) finished in 5.356 s
39033 [dag-scheduler-event-loop] INFO org.apache.spark.scheduler.DAGScheduler - Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
39034 [dag-scheduler-event-loop] INFO org.apache.spark.scheduler.cluster.YarnScheduler - Killing all running tasks in stage 0: Stage finished
39047 [main] INFO org.apache.spark.scheduler.DAGScheduler - Job 0 finished: reduce at SparkPi.scala:38, took 5.631433 s
Pi is roughly 3.1431391431391433
39073 [main] INFO org.sparkproject.jetty.server.AbstractConnector - Stopped Spark@46b695ec[HTTP/1.1,[http/1.1]]{172.28.1.1:4040}
39079 [main] INFO org.apache.spark.ui.SparkUI - Stopped Spark web UI at http://localhost:4040
39088 [YARN application state monitor] INFO org.apache.spark.scheduler.cluster.YarnClientSchedulerBackend - Interrupting monitor thread
39191 [main] INFO org.apache.spark.scheduler.cluster.YarnClientSchedulerBackend - Shutting down all executors
39197 [dispatcher-CoarseGrainedScheduler] INFO org.apache.spark.scheduler.cluster.YarnSchedulerBackend$YarnDriverEndpoint - Asking each executor to shutdown
39215 [main] INFO org.apache.spark.scheduler.cluster.YarnClientSchedulerBackend - YARN client scheduler backend Stopped
39307 [dispatcher-event-loop-3] INFO org.apache.spark.MapOutputTrackerMasterEndpoint - MapOutputTrackerMasterEndpoint stopped!
39343 [main] INFO org.apache.spark.storage.memory.MemoryStore - MemoryStore cleared
39346 [main] INFO org.apache.spark.storage.BlockManager - BlockManager stopped
39361 [main] INFO org.apache.spark.storage.BlockManagerMaster - BlockManagerMaster stopped
39369 [dispatcher-event-loop-3] INFO org.apache.spark.scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint - OutputCommitCoordinator stopped!
39419 [main] INFO org.apache.spark.SparkContext - Successfully stopped SparkContext
39434 [shutdown-hook-0] INFO org.apache.spark.util.ShutdownHookManager - Shutdown hook called
39437 [shutdown-hook-0] INFO org.apache.spark.util.ShutdownHookManager - Deleting directory /tmp/spark-42693c11-05a8-4448-92a5-462edf4d57f5
39470 [shutdown-hook-0] INFO org.apache.spark.util.ShutdownHookManager - Deleting directory /tmp/spark-cb00650f-c7a1-4461-9a54-0b773791c9ba
bash-5.0#
```

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.

[illegible]

5. Generating and Transforming Random Sentences in the Scala Shell

Generate random sentences and apply a transformation of your choice:

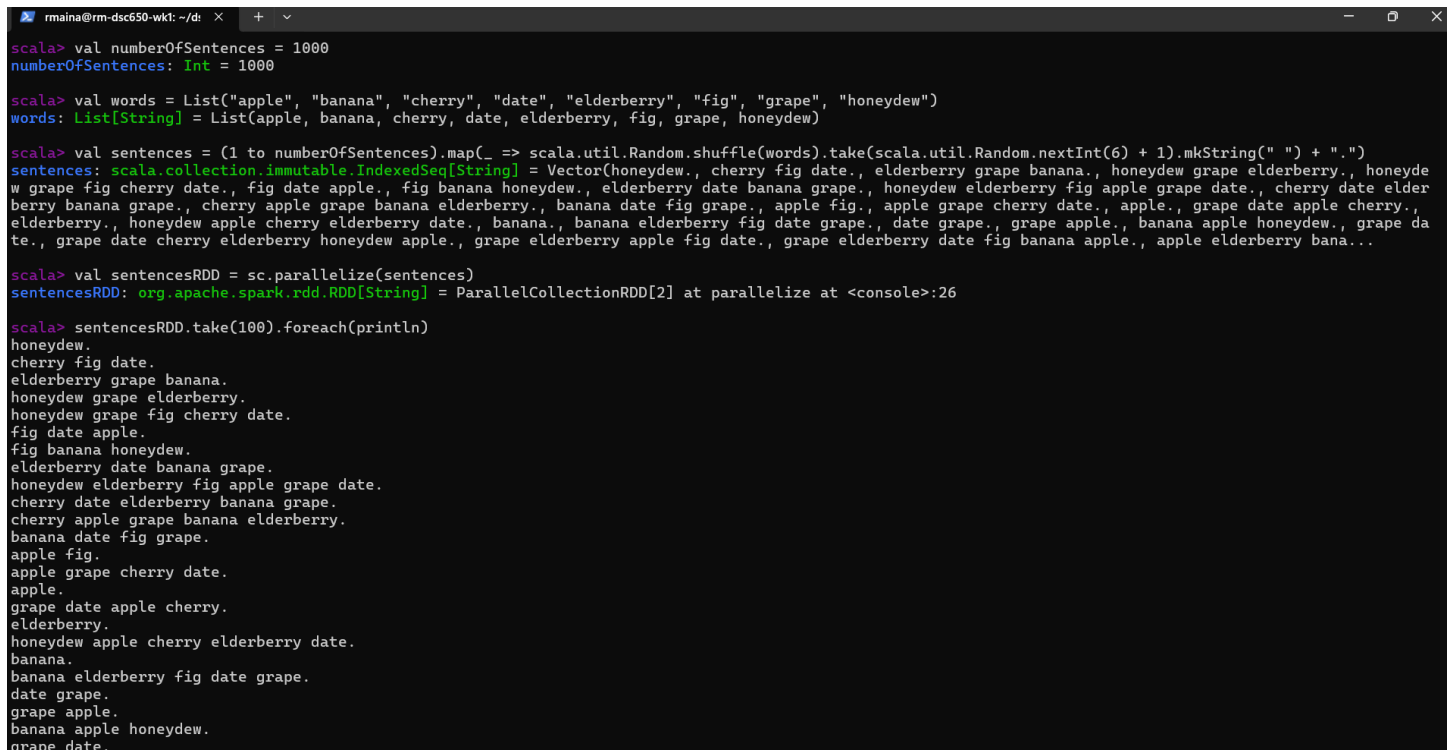
```
val numberOfSentences = 1000
val words = List("apple", "banana", "cherry", "date", "elderberry", "fig", "grape", "honeydew")
val sentences = (1 to numberOfSentences).map(_ => scala.util.Random.shuffle(words).take(scala.util.Random.nextInt(6) + 1).mkString(" ") + ".")
val sentencesRDD = sc.parallelize(sentences)

// 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:



```
rmaina@rm-dsc650-wkt: ~/dt  X + v
scala> val numberOfSentences = 1000
numberOfSentences: Int = 1000

scala> val words = List("apple", "banana", "cherry", "date", "elderberry", "fig", "grape", "honeydew")
words: List[String] = List(apple, banana, cherry, date, elderberry, fig, grape, honeydew)

scala> val sentences = (1 to numberOfSentences).map(_ => scala.util.Random.shuffle(words).take(scala.util.Random.nextInt(6) + 1).mkString(" ") + ".")
sentences: scala.collection.immutable.IndexedSeq[String] = Vector(honeydew., cherry fig date., elderberry grape banana., honeydew grape elderberry., honeyde
w grape fig cherry date., fig date apple., fig banana honeydew., elderberry date banana grape., honeydew elderberry fig apple grape date., cherry date elder
berry banana grape., cherry apple grape banana elderberry., banana date fig grape., apple fig., apple grape cherry date., apple., grape date apple cherry.,
elderberry., honeydew apple cherry elderberry date., banana., banana elderberry fig date grape., date grape., grape apple., banana apple honeydew., grape da
te., grape date cherry elderberry honeydew apple., grape elderberry apple fig date., grape elderberry date fig banana apple., apple elderberry bana...)

scala> val sentencesRDD = sc.parallelize(sentences)
sentencesRDD: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[2] at parallelize at <console>:26

scala> sentencesRDD.take(100).foreach(println)
honeydew.
cherry fig date.
elderberry grape banana.
honeydew grape elderberry.
honeydew grape fig cherry date.
fig date apple.
fig banana honeydew.
elderberry date banana grape.
honeydew elderberry fig apple grape date.
cherry date elderberry banana grape.
cherry apple grape banana elderberry.
banana date fig grape.
apple fig.
apple grape cherry date.
apple.
grape date apple cherry.
elderberry.
honeydew apple cherry elderberry date.
banana.
banana elderberry fig date grape.
date grape.
grape apple.
banana apple honeydew.
grape date.
```

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

```

scala> val transformedSentences = sentencesRDD.map(sentence => sentence.toUpperCase)
transformedSentences: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[3] at map at <console>:25

scala> transformedSentences.take(100).foreach(println)
HONEYDEW.
CHERRY FIG DATE.
ELDERBERRY GRAPE BANANA.
HONEYDEW GRAPE ELDERBERRY.
HONEYDEW GRAPE FIG CHERRY DATE.
FIG DATE APPLE.
FIG BANANA HONEYDEW.
ELDERBERRY DATE BANANA GRAPE.
HONEYDEW ELDERBERRY FIG APPLE GRAPE DATE.
CHERRY DATE ELDERBERRY BANANA GRAPE.
CHERRY APPLE GRAPE BANANA ELDERBERRY.
BANANA DATE FIG GRAPE.
APPLE FIG.
APPLE GRAPE CHERRY DATE.
APPLE.
GRAPE DATE APPLE CHERRY.
ELDERBERRY.
HONEYDEW APPLE CHERRY ELDERBERRY DATE.
BANANA.
BANANA ELDERBERRY FIG DATE GRAPE.
DATE GRAPE.
GRAPE APPLE.
BANANA APPLE HONEYDEW.
GRAPE DATE.
GRAPE DATE CHERRY ELDERBERRY HONEYDEW APPLE.
GRAPE ELDERBERRY APPLE FIG DATE.
GRAPE ELDERBERRY DATE FIG BANANA APPLE.
APPLE ELDERBERRY BANANA.
BANANA GRAPE APPLE CHERRY FIG.
BANANA GRAPE APPLE ELDERBERRY FIG.
FIG DATE GRAPE.
GRAPE BANANA HONEYDEW DATE FIG APPLE.
DATE APPLE CHERRY BANANA.
HONEYDEW CHERRY FIG.
DATE GRAPE CHERRY ELDERBERRY.
ELDERBERRY FIG.

```

```

CHERRY ELDERBERRY GRAPE.
BANANA ELDERBERRY FIG DATE GRAPE HONEYDEW.
FIG APPLE CHERRY DATE ELDERBERRY.
GRAPE APPLE.
CHERRY.
GRAPE FIG ELDERBERRY DATE APPLE CHERRY.
DATE HONEYDEW CHERRY GRAPE FIG.
GRAPE BANANA DATE APPLE.
HONEYDEW DATE GRAPE APPLE ELDERBERRY CHERRY.
HONEYDEW.
GRAPE.
DATE BANANA FIG GRAPE.
BANANA APPLE DATE ELDERBERRY.
BANANA DATE CHERRY HONEYDEW.
HONEYDEW GRAPE.
FIG APPLE HONEYDEW GRAPE.
APPLE.
APPLE GRAPE FIG HONEYDEW BANANA.
FIG CHERRY.
APPLE GRAPE ELDERBERRY FIG.
DATE.
APPLE FIG ELDERBERRY.
ELDERBERRY.
HONEYDEW.
CHERRY APPLE FIG ELDERBERRY DATE GRAPE.
DATE.
GRAPE HONEYDEW DATE BANANA CHERRY.
ELDERBERRY GRAPE APPLE CHERRY.
HONEYDEW ELDERBERRY APPLE CHERRY.
HONEYDEW.
GRAPE DATE.
HONEYDEW BANANA.
FIG.
CHERRY.
FIG GRAPE BANANA DATE.
FIG.
GRAPE FIG BANANA DATE CHERRY.
GRAPE BANANA FIG APPLE ELDERBERRY CHERRY.
FIG HONEYDEW.

scala>

```

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.