Contents

Exercise 1: Setting up the Environment

Step1: Installing Vmware from Training Bundle.

- Under 'Training Bundle' folder, navigate to "Software" folder.
- Find the executable file and complete the installation.

Step2: Extracting the Image

- From the same training bundle, locate the folder named VM image> 'cloudera-quickstart-vm-5.12.0-0-vmware'
- Load the image to get started with the machine.

- Verify all the running services with jps command

```
[cloudera@quickstart ~]$ sudo jps
6403 RESTServer
5709 SecondaryNameNode
5980 NodeManager
6878 RunJar
8543
10329 ZeppelinServer
9721 org.eclipse.equinox.launcher_1.3.0.v20140415-2008.jar
5841 Bootstrap
5366 DataNode
5454 JournalNode
8584
7516 HistoryServer
8518 Bootstrap
5573 NameNode
7484 Bootstrap
5303 QuorumPeerMain
6659 RunJar
5897 JobHistoryServer
18565 Jps
6184 ResourceManager
6521 ThriftServer
8191 Bootstrap
[cloudera@quickstart ~]$
```

Exercise 2: Practicing HDFS Commands

Task 1: Using Hadoop Command Line Interface

Navigate to Application > System Tools > Terminal

OR

Use shortcut to open the terminal.

- Operation on Files and directories

To check the content of root directory in

HDFS hdfs dfs -ls /

```
[cloudera@quickstart ~]$ hdfs dfs -ls /
Found 5 items
drwxrwxrwx   - hdfs  supergroup          0 2016-04-05 23:56 /benchmarks
drwxr-xr-x   - hbase supergroup          0 2016-09-04 10:14 /hbase
drwxrwxrwt   - hdfs  supergroup          0 2016-08-03 01:56 /tmp
drwxr-xr-x   - hdfs  supergroup          0 2016-08-02 08:08 /user
drwxr-xr-x   - hdfs  supergroup          0 2016-04-05 23:58 /var
[cloudera@quickstart ~]$
```

View the content of /user directory

hdfs dfs -ls /user

```
[cloudera@quickstart ~]$ hdfs dfs -ls /user
Found 9 items
drwxr-xr-x   - cloudera cloudera            0 2016-08-03 01:54 /user/cloudera
drwxr-xr-x   - hdfs     supergroup          0 2016-08-02 08:08 /user/hdfs
drwxr-xr-x   - mapred   hadoop              0 2016-04-05 23:56 /user/history
drwxrwxrwx   - hive     supergroup          0 2016-04-05 23:58 /user/hive
drwxrwxrwx   - hue      supergroup          0 2016-04-05 23:57 /user/hue
drwxrwxrwx   - jenkins  supergroup          0 2016-04-05 23:56 /user/jenkins
drwxrwxrwx   - oozie    supergroup          0 2016-04-05 23:57 /user/oozie
drwxrwxrwx   - root     supergroup          0 2016-04-05 23:57 /user/root
drwxr-xr-x   - hdfs     supergroup          0 2016-04-05 23:58 /user/spark
```

For an empty directory the prompt does not show any error while querying whereas if the directory doesn't exist it will throw an error.

Example: Query a non-existing directory say /music

```
hdfs dfs -ls /music
```

```
[cloudera@quickstart ~]$ hdfs dfs -ls /music
ls: `/music': No such file or directory
[cloudera@quickstart ~]$
```

The directory structure of Hadoop filesystem is different from local filesystem i.e. Linux filesystem. To know the difference, list the files on your local filesystem.

Create a new directory named "hadoop" in HDFS

```
hdfs dfs -mkdir hadoop
```

Create a sample.txt file on local and upload the file into newly created directory

```
hdfs dfs -put sample.txt hadoop/sample.txt
```

View the content of new file

```
hdfs dfs -cat hadoop/sample.txt
```

In HDFS, any non-absolute path is considered relative to your home directory i.e. /user/cloudera. Unlike Linux filesystem concept of "current" or "present working directory".

Download a file from HDFS to your local filesystem, specify HDFS path and local path to achieve the same.

```
hdfs dfs -get /user/cloudera/sample.text /home/cloudera
```

Remove the directory along with its content (perform this step after completing Task 2)

```
hdfs dfs -rm -r hadoop
```

List all the shell commands supported by Hadoop

```
hdfs dfs
```

Task2: Using HUE File browser

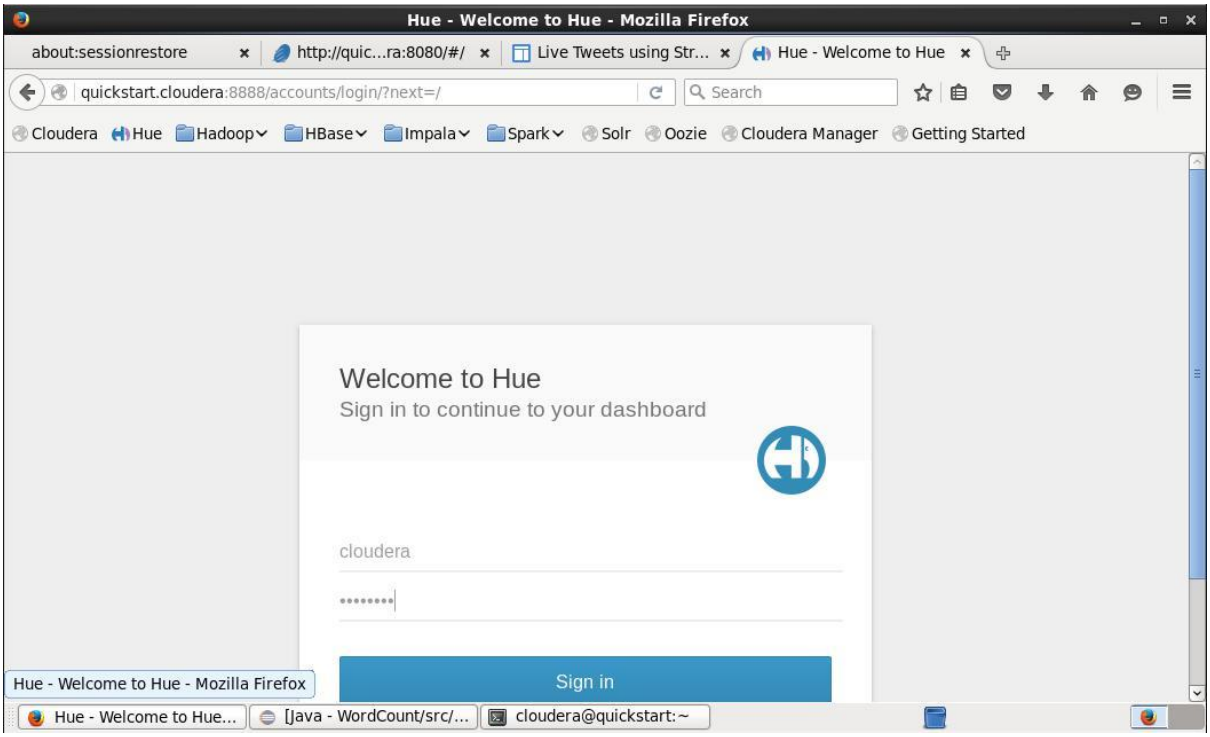
1. Open a web browser on your VM, and navigate to bookmark tab and select HUE. This can be alternatively launched by specifying http://quickstart.cloudera:8888
2. Enter username as 'cloudera' and password 'cloudera'.

3. To access HDFS, click on Manage HDFS in the HUE menu bar
4. By default, the content of home directory i.e. /user/cloudera will be visible.
5. Point out to the directory named "hadoop" created above and view the file 'sample.txt'
6. Click on Upload button and select the appropriate format of the file to be uploaded i.e. plain file or zipped file

Note

The zipped file will be automatically unzipped after upload.

7. Select Files> Select Files, and browser to location of data file on your local filesystem
8. Choose the file and click the Open button.
9. The selected file will be displayed in the directory /user/cloudera/
10. Click on the checkbox next to file's icon and then tab on Actions button to know the possible actions that can be performed on the file.
11. Once you have practised above steps, you can remove the files by clicking on "Move to Trash" button.



Exercise 3: Launching Spark

- Spark shell can be launched in two ways i.e. Scala and Python.
- To launch Scala spark shell, follow below steps

```
$ spark-shell
```

```
[cloudera@quickstart ~]$ spark-shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/
slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/St
aticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_67)
```

You will see several INFO and WARNING message on the command prompt after launching spark-shell, which can be disregarded.

Logs will appear as below and finally you will get Scala Prompt

```
SQL context available as sqlContext.

scala>
```

- Spark creates a SparkContext object called sc, verify that the object exists

```
scala> sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@a23c46d
```

- To know various SparkContext methods which are available, type sc. (sc followed by dot) and then TAB key

```
scala> sc.
accumulable                   accumulableCollection
accumulator                   addFile
addJar                        addSparkListener
appName                       applicationAttemptId
applicationId                 asInstanceOf
binaryFiles                   binaryRecords
broadcast                     cancelAllJobs
cancelJobGroup                clearCallSite
clearFiles                    clearJars
clearJobGroup                 defaultMinPartitions
defaultMinSplits              defaultParallelism
emptyRDD                      externalBlockStoreFolderName
files                         getAllPools
getCheckpointDir              getConf
getExecutorMemoryStatus       getExecutorStorageStatus
getLocalProperty              getPersistentRDDs
getPoolForName                getRDDStorageInfo
getSchedulingMode             hadoopConfiguration
hadoopFile                    hadoopRDD
initLocalProperties           isInstanceOf
isLocal                       isStopped
jars                          killExecutor
killExecutors                 makeRDD
```

Spark is based on the concept of Resilient Distributed Dataset (RDD), which is fault tolerant collection of elements that can be operated in parallel.

- Invoke python spark shell by using 'pyspark'

```
[cloudera@quickstart ~]$ pyspark
Python 2.6.6 (r266:84292, Jul 23 2015, 15:22:56)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-11)] on linux2
Type "help", "copyright", "credits" or "license" for more information.

Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Python version 2.6.6 (r266:84292, Jul 23 2015 15:22:56)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

Two ways to create RDDs:

- Parallelizing an existing collection
- Referencing a dataset from an External Storage System

Parallelized collection:

To create a parallelized collection holding numbers 1 to 6, use **sc.parallelize** method

```
val data = Array (1,2,3,4,5,6)
val distData = sc.parallelize(data)
```

Once 'distData' RDD is created, we can perform **Action** such as:

- Finding the sum, mean & variance

```
distData.sum
distData.mean
distData.variance
```

External Storage System:

Spark can create distributed datasets from any storage system supported by Hadoop, Spark supports text files, Sequence Files and any other Hadoop Input Format.

Load a file from your Local Filesystem say batting.txt using **sc.textFile** method.

```
val textFile = sc.textFile("file:/home/cloudera/datasets/story.txt")
```

Operations on RDD will be discussed in next section.

Exploring Data using RDD operations:

Transformations

filter

Return a new dataset formed by selecting those elements of the source on which *func* returns true.

```
val a = sc.parallelize(1 to 10)
val b = a.filter(_% 2 == 0)
b.collect
```

```
scala> val a = sc.parallelize(1 to 10)
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:21

scala> val b = a.filter(_% 2 == 0)
b: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[1] at filter at <console>:23

scala> b.collect
16/08/30 06:59:25 INFO spark.SparkContext: Starting job: collect at <console>:26
```

```
16/08/30 06:59:28 INFO scheduler.DAGScheduler: ResultStage 0 (collect at <console>:26) finished in 0.316 s
16/08/30 06:59:28 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 233 ms on localhost (1/1)
16/08/30 06:59:28 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
16/08/30 06:59:28 INFO scheduler.DAGScheduler: Job 0 finished: collect at <console>:26, took 2.285340 s
res0: Array[Int] = Array(2, 4, 6, 8, 10)
```

map

Return a new distributed dataset formed by passing each element of the source through a function *func*.

```
val a = sc.parallelize(List("dog", "salmon", "salmon", "rat", "elephant"))

val b = a.map(_.length)

val c = a.zip(b)

c.collect
```

```
scala> val a = sc.parallelize(List("dog", "salmon", "salmon", "rat", "elephant"))
a: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[2] at parallelize at <console>:21

scala> val b = a.map(_.length)
b: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[3] at map at <console>:23

scala> val c = a.zip(b)
c: org.apache.spark.rdd.RDD[(String, Int)] = ZippedPartitionsRDD2[4] at zip at <console>:25

scala> c.collect
16/08/30 07:03:58 INFO spark.SparkContext: Starting job: collect at <console>:28
```

```
16/08/30 07:03:58 INFO executor.Executor: Finished task 0.0 in stage 1.0 (TID 1). 1147 bytes result sent to driver
16/08/30 07:03:58 INFO scheduler.DAGScheduler: ResultStage 1 (collect at <console>:28) finished in 0.062 s
16/08/30 07:03:58 INFO scheduler.DAGScheduler: Job 1 finished: collect at <console>:28, took 0.108580 s
16/08/30 07:03:58 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 1.0 (TID 1) in 64 ms on localhost (1/1)
res1: Array[(String, Int)] = Array((dog,3), (salmon,6), (salmon,6), (rat,3), (elephant,8))
```

Using Python Shell:

```
 nums = sc.parallelize([1, 2, 3, 4])

squared = nums.map(lambda x: x * x).collect()

for num in squared: print "%i " % (num)
```

```
>>> for num in squared: print "%i " % (num)
...
1
4
9
16
>>>
```

distinct

Return a new dataset that contains the distinct elements of the source dataset.

```
val c = sc.parallelize(List("Gnu", "Cat", "Rat", "Dog", "Gnu", "Rat"),

2) . c.distinct.collect
```

```
scala> val c = sc.parallelize(List("Gnu", "Cat", "Rat", "Dog", "Gnu", "Rat"), 2)
c: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[5] at parallelize at <console>:21

scala> c.distinct.collect
16/08/30 07:07:16 INFO spark.SparkContext: Starting job: collect at <console>:24
```

```
16/08/30 07:07:19 INFO executor.Executor: Finished task 1.0 in stage 3.0 (TID 5). 1171 bytes result sent to driver
16/08/30 07:07:19 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 3.0 (TID 5) in 76 ms on localhost (2/2)
16/08/30 07:07:19 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have all completed, from pool
16/08/30 07:07:19 INFO scheduler.DAGScheduler: ResultStage 3 (collect at <console>:24) finished in 0.385 s
16/08/30 07:07:19 INFO scheduler.DAGScheduler: Job 2 finished: collect at <console>:24, took 2.185199 s
res2: Array[String] = Array(Dog, Cat, Gnu, Rat)
```

cartesian

When called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements).

```
val x = sc.parallelize(List(1,2,3,4,5)) .

val y = sc.parallelize(List(6,7,8,9,10)) .

x.cartesian(y).collect
```

```
scala> val x = sc.parallelize(List(1,2,3,4,5))
x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[9] at parallelize at <console>:21

scala> val y = sc.parallelize(List(6,7,8,9,10))
y: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[10] at parallelize at <console>:21

scala> x.cartesian(y).collect
16/08/30 07:10:11 INFO spark.SparkContext: Starting job: collect at <console>:26
```

```
16/08/30 07:10:11 INFO scheduler.DAGScheduler: ResultStage 4 (collect at <console>:26) finished in 0.145 s
16/08/30 07:10:11 INFO scheduler.DAGScheduler: Job 3 finished: collect at <console>:26, took 0.264577 s
16/08/30 07:10:11 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 4.0 (TID 6) in 153 ms on localhost (1/1)
16/08/30 07:10:11 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 4.0, whose tasks have all completed, from pool
res3: Array[(Int, Int)] = Array((1,6), (1,7), (1,8), (1,9), (1,10), (2,6), (2,7), (2,8), (2,9), (2,10), (3,6), (3,7), (3,8), (3,
9), (3,10), (4,6), (4,7), (4,8), (4,9), (4,10), (5,6), (5,7), (5,8), (5,9), (5,10))
```

coalesce

Decrease the number of partitions in the RDD to numPartitions. Useful for running operations more efficiently after filtering down a large dataset.

```
val y = sc.parallelize(1 to 10, 10)

val z = y.coalesce(2, false)

z.partitions.length
```

```
scala> val y = sc.parallelize(1 to 10, 10)
y: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[12] at parallelize at <console>:21

scala> val z = y.coalesce(2, false)
z: org.apache.spark.rdd.RDD[Int] = CoalescedRDD[13] at coalesce at <console>:23

scala> z.partitions.length
res4: Int = 2
```

filterByRange

Returns an RDD containing only the items in the key range specified.

```
val randRDD = sc.parallelize(List( (2,"cat"), (6, "mouse"),(7, "cup"),
(3, "book"), (4, "tv"), (1, "screen"), (5, "heater")), 3)

val sortedRDD = randRDD.sortByKey()

sortedRDD.filterByRange(1,3).collect
```

```
16/08/30 07:14:40 INFO executor.Executor: Finished task 0.0 in stage 7.0 (TID 13). 1357 bytes result sent to driver
16/08/30 07:14:40 INFO scheduler.DAGScheduler: ResultStage 7 (collect at <console>:26) finished in 0.080 s
16/08/30 07:14:40 INFO scheduler.DAGScheduler: Job 5 finished: collect at <console>:26, took 0.527731 s
res5: Array[(Int, String)] = Array((1,screen), (2,cat), (3,book))
```

flatMap

Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).

```
val a = sc.parallelize(1 to 10, 5)

a.flatMap(1 to _).collect
```

```
16/08/30 07:18:44 INFO executor.Executor: Finished task 4.0 in stage 9.0 (TID 20). 974 bytes result sent to
16/08/30 07:18:44 INFO scheduler.TaskSetManager: Finished task 4.0 in stage 9.0 (TID 20) in 24 ms on localho
16/08/30 07:18:44 INFO scheduler.DAGScheduler: ResultStage 9 (collect at <console>:24) finished in 0.141 s
16/08/30 07:18:44 INFO scheduler.DAGScheduler: Job 7 finished: collect at <console>:24, took 0.193634 s
res6: Array[Int] = Array(1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 7,
, 8, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
val b = sc.parallelize(List(1, 2, 3), 2).flatMap(x => List(x,
x, x)).collect
res85: Array[Int] = Array(1, 1, 1, 2, 2, 2, 3, 3, 3)
```

```
16/08/30 07:17:29 INFO executor.Executor: Running task 1.0 in stage 8.0 (TID 15)
16/08/30 07:17:29 INFO executor.Executor: Finished task 1.0 in stage 8.0 (TID 15). 922 bytes result sent to driver
16/08/30 07:17:29 INFO scheduler.DAGScheduler: ResultStage 8 (collect at <console>:23) finished in 0.104 s
16/08/30 07:17:29 INFO scheduler.DAGScheduler: Job 6 finished: collect at <console>:23, took 0.126929 s
b: Array[Int] = Array(1, 1, 1, 2, 2, 2, 3, 3, 3)
```

```
val lines = sc.parallelize(List("hello world",
"hi")) val words = lines.flatMap(line =>
line.split(" ")) words.first() // returns "hello"
```

```
16/08/30 07:21:01 INFO scheduler.DAGScheduler: ResultStage 10 (first at <console>:26) finished in 0.006 s
16/08/30 07:21:01 INFO scheduler.DAGScheduler: Job 8 finished: first at <console>:26, took 0.035576 s
res7: String = hello
```

Using python shell:

```
lines = sc.parallelize(["hello world", "hi"])
words = lines.flatMap(lambda line: line.split(" "))
words.first()  # returns "hello"
```

```
>>> lines = sc.parallelize(["hello world", "hi"])
>>> words = lines.flatMap(lambda line: line.split(" "))
>>> words.first()
16/09/17 09:38:59 INFO spark.SparkContext: Starting job: runJob at PythonRDD.sca
la:393

'hello'
>>>
```

groupBy

```
val a = sc.parallelize(1 to 9, 3)
a.groupBy(x => { if (x % 2 == 0) "even" else "odd" }).collect
```

```
16/08/30 07:23:14 INFO scheduler.DAGScheduler: ResultStage 12 (collect at <console>:26) finished in 0.182 s
16/08/30 07:23:14 INFO scheduler.DAGScheduler: Job 9 finished: collect at <console>:26, took 0.447740 s
res8: Array[(String, Iterable[Int])] = Array((even,CompactBuffer(2, 4, 6, 8)), (odd,CompactBuffer(1, 3, 5, 7
, 9)))
```

keys

Extracts the keys from all contained tuples and returns them in a new RDD.

```
val a = sc.parallelize(List("dog", "tiger", "lion", "cat",
"panther", "eagle"), 2)
val b = a.map(x => (x.length, x))
b.keys.collect
```

```
16/08/30 07:25:04 INFO scheduler.DAGScheduler: ResultStage 13 (collect at <console>:28) finished in 0.089 s
16/08/30 07:25:04 INFO scheduler.DAGScheduler: Job 10 finished: collect at <console>:28, took 0.131419 s
res9: Array[Int] = Array(3, 5, 4, 3, 7, 5)
```

union

```
val seta = sc.parallelize(1 to 10)
val setb = sc.parallelize(5 to 15)
(seta union setb).collect
```

```
16/08/30 07:26:33 INFO scheduler.DAGScheduler: ResultStage 14 (collect at <console>:26) finished in 0.143 s
16/08/30 07:26:33 INFO scheduler.DAGScheduler: Job 11 finished: collect at <console>:26, took 0.318252 s
res10: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)
```

zip

Joins two RDDs by combining the i-th of either partition with each other. The resulting RDD will consist of two-component tuples which are interpreted as key-value pairs by the methods provided by the PairRDDFunctions extension.

```
val a = sc.parallelize(1 to 100, 3)
val b = sc.parallelize(101 to 200, 3)
a.zip(b).collect
```

```
16/08/30 07:30:02 INFO scheduler.TaskSetManager: Finished task 2.0 in stage 15.0 (TID 34) in 35 ms on localh
ost (3/3)
16/08/30 07:30:02 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 15.0, whose tasks have all completed, fr
om pool
res11: Array[(Int, Int)] = Array((1,101), (2,102), (3,103), (4,104), (5,105), (6,106), (7,107), (8,108), (9,
109), (10,110), (11,111), (12,112), (13,113), (14,114), (15,115), (16,116), (17,117), (18,118), (19,119), (2
0,120), (21,121), (22,122), (23,123), (24,124), (25,125), (26,126), (27,127), (28,128), (29,129), (30,130),
(31,131), (32,132), (33,133), (34,134), (35,135), (36,136), (37,137), (38,138), (39,139), (40,140), (41,141)
, (42,142), (43,143), (44,144), (45,145), (46,146), (47,147), (48,148), (49,149), (50,150), (51,151), (52,15
2), (53,153), (54,154), (55,155), (56,156), (57,157), (58,158), (59,159), (60,160), (61,161), (62,162), (63,
163), (64,164), (65,165), (66,166), (67,167), (68,168), (69,169), (70,170), (71,171), (72,172), (73,173), (7
4,174), (75,175), (76,176), (77,177), (78...
scala>
```

Action

collect

Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.

```
val c = sc.parallelize(List("Gnu", "Cat", "Rat", "Dog", "Gnu", "Rat"),
2) c.collect
```

```
16/08/30 07:31:52 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 16.0 (TID 36) in 7 ms on localho
st (2/2)
16/08/30 07:31:52 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 16.0, whose tasks have all completed, fr
om pool
16/08/30 07:31:52 INFO scheduler.DAGScheduler: ResultStage 16 (collect at <console>:24) finished in 0.010 s
16/08/30 07:31:52 INFO scheduler.DAGScheduler: Job 13 finished: collect at <console>:24, took 0.018261 s
res12: Array[String] = Array(Gnu, Cat, Rat, Dog, Gnu, Rat)
```

collectAsMap

Similar to collect, but works on key-value RDDs and converts them into Scala maps to preserve their key-value structure.

```
val a = sc.parallelize(List(1, 2, 1, 3), 1)

val b = a.zip(a)

b.collectAsMap
```

```
16/08/30 07:33:33 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 17.0 (TID 37) in 145 ms on local
host (1/1)
16/08/30 07:33:33 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 17.0, whose tasks have all completed, fr
om pool
res13: scala.collection.Map[Int,Int] = Map(2 -> 2, 1 -> 1, 3 -> 3)
```

count

Return the number of elements in the dataset.

```
val c = sc.parallelize(List("Gnu", "Cat", "Rat", "Dog"), 2)

c.count

res2: Long = 4
```

```
16/08/30 07:34:52 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 18.0 (TID 39) in 11 ms on localh
ost (2/2)
16/08/30 07:34:52 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 18.0, whose tasks have all completed, fr
om pool
16/08/30 07:34:52 INFO scheduler.DAGScheduler: ResultStage 18 (count at <console>:24) finished in 0.040 s
16/08/30 07:34:52 INFO scheduler.DAGScheduler: Job 15 finished: count at <console>:24, took 0.054256 s
res14: Long = 4
```

reduce

Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.

```
val a = sc.parallelize(1 to 100, 3)

a.reduce(_+_)
```

```
16/08/30 08:32:49 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 19.0, whose tasks have all completed, fr
om pool
16/08/30 08:32:49 INFO scheduler.DAGScheduler: ResultStage 19 (reduce at <console>:24) finished in 0.099 s
16/08/30 08:32:49 INFO scheduler.DAGScheduler: Job 16 finished: reduce at <console>:24, took 0.315500 s
res15: Int = 5050
```

take

Return an array with the first *n* elements of the dataset.

```
val b = sc.parallelize(List("dog", "cat", "ape", "salmon", "gnu"), 2)

b.take(2)
```

```
16/08/30 08:34:15 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 20.0, whose tasks have all completed, fr
om pool
16/08/30 08:34:15 INFO scheduler.DAGScheduler: ResultStage 20 (take at <console>:24) finished in 0.035 s
16/08/30 08:34:15 INFO scheduler.DAGScheduler: Job 17 finished: take at <console>:24, took 0.142047 s
res16: Array[String] = Array(dog, cat)
```

```
val b = sc.parallelize(1 to 100, 5)

b.take(30)
```

```
16/08/30 08:35:57 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 22.0, whose tasks have all completed, fr
om pool
16/08/30 08:35:58 INFO scheduler.DAGScheduler: ResultStage 22 (take at <console>:24) finished in 0.004 s
16/08/30 08:35:58 INFO scheduler.DAGScheduler: Job 19 finished: take at <console>:24, took 0.064874 s
res17: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
 24, 25, 26, 27, 28, 29, 30)
```

first

Return the first element of the dataset (similar to take(1)).

```
val c = sc.parallelize(List("Gnu", "Cat", "Rat", "Dog"), 2)

c.first
```

```
16/08/30 08:37:15 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 23.0, whose tasks have all completed, fr
om pool
16/08/30 08:37:15 INFO scheduler.DAGScheduler: ResultStage 23 (first at <console>:24) finished in 0.006 s
16/08/30 08:37:15 INFO scheduler.DAGScheduler: Job 20 finished: first at <console>:24, took 0.016040 s
res18: String = Gnu
```

countByValue

Returns a map that contains all unique values of the RDD and their respective occurrence counts.

```
val b = sc.parallelize(List(1,2,3,4,5,6,7,8,2,4,2,1,1,1,1,1))

b.countByValue
```

```
16/08/30 08:38:18 INFO scheduler.DAGScheduler: ResultStage 25 (countByValue at <console>:24) finished in 0.0
40 s
16/08/30 08:38:18 INFO scheduler.DAGScheduler: Job 21 finished: countByValue at <console>:24, took 0.495353
s
res19: scala.collection.Map[Int,Long] = Map(5 -> 1, 1 -> 6, 6 -> 1, 2 -> 3, 7 -> 1, 3 -> 1, 8 -> 1, 4 -> 2)
```

lookup

Scans the RDD for all keys that match the provided value and returns their values as a Scala sequence.

```
val a = sc.parallelize(List("dog", "tiger", "lion", "cat",
"panther", "eagle"), 2)
val b = a.map(x => (x.length, x))

b.lookup(5)
```

```
16/08/30 08:39:47 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 26.0, whose tasks have all completed, fr
om pool
16/08/30 08:39:47 INFO scheduler.DAGScheduler: ResultStage 26 (lookup at <console>:28) finished in 0.087 s
16/08/30 08:39:47 INFO scheduler.DAGScheduler: Job 22 finished: lookup at <console>:28, took 0.188596 s
res20: Seq[String] = WrappedArray(tiger, eagle)
```

max

Returns the largest element in the RDD

```
val y = sc.parallelize(10 to 30)

y.max
```

```
16/08/30 08:41:08 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 27.0, whose tasks have all completed, fr
om pool
16/08/30 08:41:08 INFO scheduler.DAGScheduler: ResultStage 27 (max at <console>:24) finished in 0.042 s
16/08/30 08:41:08 INFO scheduler.DAGScheduler: Job 23 finished: max at <console>:24, took 0.115601 s
res21: Int = 30
```

```
val a = sc.parallelize(List((10, "dog"), (3, "tiger"), (9, "lion"),
(18, "cat")))
a.max
```

```
16/08/30 08:42:29 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 28.0, whose tasks have all completed, fr
om pool
16/08/30 08:42:29 INFO scheduler.DAGScheduler: ResultStage 28 (max at <console>:24) finished in 0.031 s
16/08/30 08:42:29 INFO scheduler.DAGScheduler: Job 24 finished: max at <console>:24, took 0.054414 s
res22: (Int, String) = (18,cat)
```

min

Returns the smallest element in the RDD

```
val y = sc.parallelize(10 to 30)
y.min
```

```
16/08/30 08:43:49 INFO executor.Executor: Finished task 0.0 in stage 29.0 (TID 53). 1031 bytes result sent t
o driver
16/08/30 08:43:49 INFO scheduler.DAGScheduler: ResultStage 29 (min at <console>:24) finished in 0.075 s
16/08/30 08:43:49 INFO scheduler.DAGScheduler: Job 25 finished: min at <console>:24, took 0.170125 s
res23: Int = 10
```

```
val a = sc.parallelize(List((10, "dog"), (3, "tiger"), (9, "lion"),
(8, "cat")))
a.min
```

```
16/08/30 08:45:21 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 30.0, whose tasks have all completed, fr
om pool
16/08/30 08:45:21 INFO scheduler.DAGScheduler: ResultStage 30 (min at <console>:24) finished in 0.012 s
16/08/30 08:45:21 INFO scheduler.DAGScheduler: Job 26 finished: min at <console>:24, took 0.020695 s
res24: (Int, String) = (3,tiger)
```

mean

Calls stats and extracts the mean component.

```
val a = sc.parallelize(List(9.1, 1.0, 1.2, 2.1, 1.3, 5.0, 2.0, 2.1,
7.4, 7.5, 7.6, 8.8, 10.0, 8.9, 5.5), 3)
a.mean
```

```
16/08/30 08:46:49 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 31.0, whose tasks have all completed, fr
om pool
16/08/30 08:46:49 INFO scheduler.DAGScheduler: ResultStage 31 (mean at <console>:24) finished in 0.425 s
16/08/30 08:46:49 INFO scheduler.DAGScheduler: Job 27 finished: mean at <console>:24, took 0.526866 s
res25: Double = 5.3
```

variance

Calls stats and extracts either variance-component or corrected sampleVariance-component.

```
val a = sc.parallelize(List(9.1, 1.0, 1.2, 2.1, 1.3, 5.0, 2.0, 2.1, 7.4,
7.5, 7.6, 8.8, 10.0, 8.9, 5.5), 3)
a.variance
```

```
16/08/30 08:48:05 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 32.0, whose tasks have all completed, fr
om pool
16/08/30 08:48:05 INFO scheduler.DAGScheduler: ResultStage 32 (variance at <console>:24) finished in 0.018 s
16/08/30 08:48:05 INFO scheduler.DAGScheduler: Job 28 finished: variance at <console>:24, took 0.032808 s
res26: Double = 10.605333333333332
```

PairRDD

countByKey

Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.

```
val c = sc.parallelize(List((3, "Gnu"), (3, "Yak"), (5, "Mouse"),
(3, "Dog")), 2)
c.countByKey
```

```
16/08/30 08:49:39 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 34.0, whose tasks have all completed, fr
om pool
16/08/30 08:49:39 INFO scheduler.DAGScheduler: ResultStage 34 (countByKey at <console>:24) finished in 0.073
 s
16/08/30 08:49:39 INFO scheduler.DAGScheduler: Job 29 finished: countByKey at <console>:24, took 0.384196 s
res27: scala.collection.Map[Int,Long] = Map(3 -> 3, 5 -> 1)
```

groupByKey

When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs.

```
val a = sc.parallelize(List("dog", "tiger", "lion", "cat",
"spider", "eagle"), 2)
val b = a.keyBy(_.length)
b.groupByKey.collect
```

```
16/08/30 08:51:02 INFO executor.Executor: Finished task 1.0 in stage 36.0 (TID 68). 1703 bytes result sent t
o driver
16/08/30 08:51:02 INFO scheduler.DAGScheduler: ResultStage 36 (collect at <console>:26) finished in 0.042 s
16/08/30 08:51:02 INFO scheduler.DAGScheduler: Job 30 finished: collect at <console>:26, took 0.415007 s
res28: Array[(Int, Iterable[String])] = Array((4,CompactBuffer(lion)), (6,CompactBuffer(spider)), (3,Compact
Buffer(dog, cat)), (5,CompactBuffer(tiger, eagle)))
```

reduceByKey

When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V,V) => V

```
val a = sc.parallelize(List("dog", "cat", "owl", "gnu", "ant"),

2) , val b = a.map(x => (x.length, x)) , b.reduceByKey(_ +

).collect
```

```
16/08/30 08:52:58 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 38.0 (TID 72) in 25 ms on localh
ost (2/2)
16/08/30 08:52:58 INFO scheduler.DAGScheduler: ResultStage 38 (collect at <console>:28) finished in 0.033 s
16/08/30 08:52:58 INFO scheduler.DAGScheduler: Job 31 finished: collect at <console>:28, took 0.413629 s
res29: Array[(Int, String)] = Array((3,dogcatowlgnuant))
```

foldByKey

Very similar to *fold*, but performs the folding separately for each key of the RDD. This function is only available if the RDD consists of two-component tuples.

```
val deptEmployees =
  List
  (
   ("dept1",("kumar1",1000.0)),
   ("dept1",("kumar2",1200.0)),
   ("dept2",("kumar3",2200.0)),
   ("dept2",("kumar4",1400.0)),
   ("dept2",("kumar5",1000.0)),
   ("dept2",("kumar6",800.0)),
   ("dept1",("kumar7",2000.0)),
   ("dept1",("kumar8",1000.0)),
   ("dept1",("kumar9",500.0))
   )


val employeeRDD = sc.makeRDD(deptEmployees)


val maxByDept =
employeeRDD.foldByKey(("dummy",Double.MinValue))((acc,element) => if(acc._2
> element._2)acc else element)

println("Maximum salaries in each dept" + maxByDept.collect().toList)
```

```
16/08/30 08:56:35 INFO executor.Executor: Finished task 0.0 in stage 40.0 (TID 74). 1375 bytes result sent t
o driver
16/08/30 08:56:35 INFO scheduler.DAGScheduler: ResultStage 40 (collect at <console>:28) finished in 0.087 s
16/08/30 08:56:35 INFO scheduler.DAGScheduler: Job 32 finished: collect at <console>:28, took 0.351474 s
Maximum salaries in each deptList((dept2,(kumar3,2200.0)), (dept1,(kumar7,2000.0)))
```

cogroup

When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (Iterable<V>, Iterable<W>)) tuples.

```
val a = sc.parallelize(List(1, 2, 1, 3),
1) . val b = a.map((_, "b")) . val c =
a.map((_, "c")) . b.cogroup(c).collect
```

```
16/08/30 08:58:50 INFO executor.Executor: Finished task 0.0 in stage 43.0 (TID 77). 2177 bytes result sent t
o driver
16/08/30 08:58:50 INFO scheduler.DAGScheduler: ResultStage 43 (collect at <console>:28) finished in 0.249 s
16/08/30 08:58:50 INFO scheduler.DAGScheduler: Job 33 finished: collect at <console>:28, took 0.668327 s
res31: Array[(Int, (Iterable[String], Iterable[String]))] = Array((1,(CompactBuffer(b, b),CompactBuffer(c, c
))), (3,(CompactBuffer(b),CompactBuffer(c))), (2,(CompactBuffer(b),CompactBuffer(c))))
```

```
val x = sc.parallelize(List((1, "apple"), (2, "banana"), (3, "orange"),
(4, . "kiwi")), 2)
val y = sc.parallelize(List((5, "computer"), (1, "laptop"), (1,
"desktop"), . (4, "iPad")), 2)
x.cogroup(y).collect
```

```
16/08/30 09:00:55 INFO executor.Executor: Finished task 1.0 in stage 46.0 (TID 83). 2189 bytes result sent t
o driver
16/08/30 09:00:55 INFO scheduler.DAGScheduler: ResultStage 46 (collect at <console>:26) finished in 0.123 s
16/08/30 09:00:55 INFO scheduler.DAGScheduler: Job 34 finished: collect at <console>:26, took 0.624333 s
res32: Array[(Int, (Iterable[String], Iterable[String]))] = Array((4,(CompactBuffer(kiwi),CompactBuffer(iPad
))), (2,(CompactBuffer(banana),CompactBuffer())), (1,(CompactBuffer(apple),CompactBuffer(laptop, desktop))),
 (3,(CompactBuffer(orange),CompactBuffer())), (5,(CompactBuffer(),CompactBuffer(computer))))
```

Join

Performs an inner join using two key-value RDDs.

```
val a = sc.parallelize(List("dog", "salmon", "salmon", "rat",
"elephant"), . 3)
val b = a.keyBy(_.length)
```

```
val c =
sc.parallelize(List("dog","cat","gnu","salmon","rabbit","turkey","wolf","be
ar","bee"), 3)
val d = c.keyBy(_.length)
b.join(d).collect
```

```
16/08/30 09:02:41 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 49.0, whose tasks have all completed, fr
om pool
16/08/30 09:02:41 INFO scheduler.DAGScheduler: ResultStage 49 (collect at <console>:30) finished in 0.470 s
16/08/30 09:02:41 INFO scheduler.DAGScheduler: Job 35 finished: collect at <console>:30, took 1.019005 s
res33: Array[(Int, (String, String))] = Array((6,(salmon,salmon)), (6,(salmon,rabbit)), (6,(salmon,turkey)),
 (6,(salmon,salmon)), (6,(salmon,rabbit)), (6,(salmon,turkey)), (3,(dog,dog)), (3,(dog,cat)), (3,(dog,gnu)),
 (3,(dog,bee)), (3,(rat,dog)), (3,(rat,cat)), (3,(rat,gnu)), (3,(rat,bee)))
```

leftOuterJoin

Performs an left outer join using two key-value RDDs.

```
val a = sc.parallelize(List("dog", "salmon", "salmon", "rat",
"elephant"), 3)
val b = a.keyBy(_.length)
val c =
sc.parallelize(List("dog","cat","gnu","salmon","rabbit","turkey","wolf","be
ar","bee"), 3)
val d = c.keyBy(_.length)
b.leftOuterJoin(d).collect
```

```
16/08/30 09:03:59 INFO scheduler.DAGScheduler: Job 36 finished: collect at <console>:30, took 0.675634 s
16/08/30 09:03:59 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 52.0, whose tasks have all completed, fr
om pool
res34: Array[(Int, (String, Option[String]))] = Array((6,(salmon,Some(salmon))), (6,(salmon,Some(rabbit))),
(6,(salmon,Some(turkey))), (6,(salmon,Some(salmon))), (6,(salmon,Some(rabbit))), (6,(salmon,Some(turkey))),
(3,(dog,Some(dog))), (3,(dog,Some(cat))), (3,(dog,Some(gnu))), (3,(dog,Some(bee))), (3,(rat,Some(dog))), (3,
(rat,Some(cat))), (3,(rat,Some(gnu))), (3,(rat,Some(bee))), (8,(elephant,None))
```

rightOuterJoin

Performs an right outer join using two key-value RDDs.

```
val a = sc.parallelize(List("dog", "salmon", "salmon", "rat",
"elephant"), 3)
val b = a.keyBy(_.length)
val c =
sc.parallelize(List("dog","cat","gnu","salmon","rabbit","turkey","wolf","be
ar","bee"), 3)
val d = c.keyBy(_.length)
```

```
b.rightOuterJoin(d).collect
```

```
16/08/30 09:05:30 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 55.0, whose tasks have all completed, fr
om pool
16/08/30 09:05:30 INFO scheduler.DAGScheduler: ResultStage 55 (collect at <console>:30) finished in 0.181 s
16/08/30 09:05:30 INFO scheduler.DAGScheduler: Job 37 finished: collect at <console>:30, took 1.003812 s
res35: Array[(Int, (Option[String], String))] = Array((6,(Some(salmon),salmon)), (6,(Some(salmon),rabbit)),
(6,(Some(salmon),turkey)), (6,(Some(salmon),salmon)), (6,(Some(salmon),rabbit)), (6,(Some(salmon),turkey)),
(3,(Some(dog),dog)), (3,(Some(dog),cat)), (3,(Some(dog),gnu)), (3,(Some(dog),bee)), (3,(Some(rat),dog)), (3,
(Some(rat),cat)), (3,(Some(rat),gnu)), (3,(Some(rat),bee)), (4,(None,wolf)), (4,(None,bear)))
```

keyBy

Constructs two-component tuples (key-value pairs) by applying a function on each data item. The result of the function becomes the key and the original data item becomes the value of the newly created tuples.

```
val a = sc.parallelize(List("dog", "salmon", "salmon", "rat",
"elephant"), 3)
val b = a.keyBy(_.length)
b.collect
```

```
16/08/30 09:14:29 INFO scheduler.TaskSetManager: Finished task 2.0 in stage 56.0 (TID 113) in 7 ms on localh
ost (3/3)
16/08/30 09:14:29 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 56.0, whose tasks have all completed, fr
om pool
16/08/30 09:14:29 INFO scheduler.DAGScheduler: ResultStage 56 (collect at <console>:26) finished in 0.053 s
16/08/30 09:14:29 INFO scheduler.DAGScheduler: Job 38 finished: collect at <console>:26, took 0.073531 s
res36: Array[(Int, String)] = Array((3,dog), (6,salmon), (6,salmon), (3,rat), (8,elephant))
```

mapValues

Takes the values of a RDD that consists of two-component tuples, and applies the provided function to transform each value. Then, it forms new two-component tuples using the key and the transformed value and stores them in a new RDD.

```
val a = sc.parallelize(List("dog", "tiger", "lion", "cat",
"panther", "eagle"), 2)
val b = a.map(x => (x.length, x))
b.mapValues("x" + _ + "x").collect
```

```
16/08/30 09:21:21 INFO executor.Executor: Finished task 1.0 in stage 57.0 (TID 115). 1118 bytes result sent
to driver
16/08/30 09:21:21 INFO scheduler.DAGScheduler: ResultStage 57 (collect at <console>:28) finished in 0.082 s
16/08/30 09:21:21 INFO scheduler.DAGScheduler: Job 39 finished: collect at <console>:28, took 0.182411 s
res37: Array[(Int, String)] = Array((3,xdogx), (5,xtigerx), (4,xlionx), (3,xcatx), (7,xpantherx), (5,xeaglex
))
```

cache

The cache( ) method is a shorthand for using the default storage level, which is StorageLevel.MEMORY_ONLY (store deserialized objects in memory)

```
val c = sc.parallelize(List("Gnu", "Cat", "Rat", "Dog", "Gnu", "Rat"),
2) c.getStorageLevel
```

```
scala> val c = sc.parallelize(List("Gnu", "Cat", "Rat", "Dog", "Gnu", "Rat"), 2)
c: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[99] at parallelize at <console>:21

scala> c.getStorageLevel
res38: org.apache.spark.storage.StorageLevel = StorageLevel(false, false, false, false, 1)
```

```
c.cache

c.getStorageLevel
```

```
scala> c.cache
res39: c.type = ParallelCollectionRDD[99] at parallelize at <console>:21

scala> c.getStorageLevel
res40: org.apache.spark.storage.StorageLevel = StorageLevel(false, true, false, true, 1)
```

repartition

Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it across them. This always shuffles all data over the network.

```
val rdd = sc.parallelize(List(1, 2, 10, 4, 5, 2, 1, 1, 1), 3)

rdd.partitions.length

val rdd2 = rdd.repartition(5)

rdd2.partitions.length
```

```
scala> val rdd = sc.parallelize(List(1, 2, 10, 4, 5, 2, 1, 1, 1), 3)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[102] at parallelize at <console>:21

scala> rdd.partitions.length
res43: Int = 3

scala> val rdd2 = rdd.repartition(5)
rdd2: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[106] at repartition at <console>:23

scala> rdd2.partitions.length
res44: Int = 5
```

Developing with Spark

REPL

Example: Counting the occurrence of lines having a particular word in it

```
val textFile = sc.textFile("file:/home/cloudera/datasets/Joyce.txt")
```

```
textFile.count() //Return the number of elements in the dataset
```

```
16/08/30 09:34:43 INFO executor.Executor: Finished task 0.0 in stage 59.0 (TID 136). 2082 bytes result sent
to driver
16/08/30 09:34:43 INFO scheduler.DAGScheduler: ResultStage 59 (count at <console>:24) finished in 2.238 s
16/08/30 09:34:43 INFO scheduler.DAGScheduler: Job 41 finished: count at <console>:24, took 2.310447 s
res45: Long = 33056
```

```
textFile.first() //Return the first element of the dataset
```

```
16/08/30 09:35:03 INFO executor.Executor: Finished task 0.0 in stage 60.0 (TID 137). 2101 bytes result sent
to driver
16/08/30 09:35:03 INFO scheduler.DAGScheduler: ResultStage 60 (first at <console>:24) finished in 0.035 s
16/08/30 09:35:03 INFO scheduler.DAGScheduler: Job 42 finished: first at <console>:24, took 0.091601 s
res46: String = The Project Gutenberg EBook of Ulysses, by James Joyce
```

```
val linesWithJoyce = textFile.filter(line => line.contains("Joyce"))

linesWithJoyce.count() //How many lines contains "Joyce"
```

```
16/08/30 09:35:26 INFO executor.Executor: Finished task 0.0 in stage 61.0 (TID 138). 2082 bytes result sent
to driver
16/08/30 09:35:26 INFO scheduler.DAGScheduler: ResultStage 61 (count at <console>:26) finished in 0.064 s
16/08/30 09:35:26 INFO scheduler.DAGScheduler: Job 43 finished: count at <console>:26, took 0.094628 s
res47: Long = 4
```

Example: Add up the sizes of all the lines

```
val lineLength = textFile.map(s => s.length)

val totalLength = lineLength.reduce((a, b) => a + b) //Aggregate the
elements of the dataset using a function
```

```
16/08/30 09:40:54 INFO executor.Executor: Finished task 0.0 in stage 62.0 (TID 139). 2160 bytes result sent
to driver
16/08/30 09:40:54 INFO scheduler.DAGScheduler: ResultStage 62 (reduce at <console>:31) finished in 0.262 s
16/08/30 09:40:54 INFO scheduler.DAGScheduler: Job 44 finished: reduce at <console>:31, took 0.337723 s
totalLength: Int = 1506967
```

Example: To find out the line with maximum words

```
textFile.map(line => line.split(" ").size).reduce((a,b) => if(a>b) a
else , b)
```

```
16/08/30 09:41:48 INFO executor.Executor: Finished task 0.0 in stage 63.0 (TID 140). 2160 bytes result sent
to driver
16/08/30 09:41:48 INFO scheduler.DAGScheduler: ResultStage 63 (reduce at <console>:30) finished in 1.048 s
16/08/30 09:41:48 INFO scheduler.DAGScheduler: Job 45 finished: reduce at <console>:30, took 1.070554 s
res48: Int = 22
```

Zeppelein

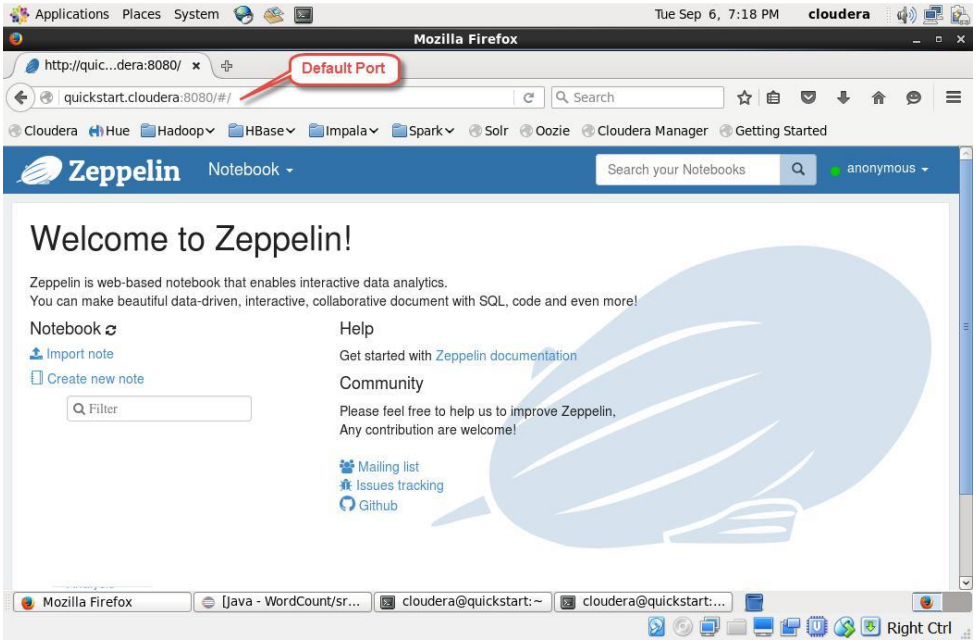
To start Zeepelin service

```
cd /usr/lib/zeppelin-0.6.0-bin-all

sudo bin/zeppelin-daemon.sh start
```

```
[cloudera@quickstart ~]$ cd /usr/lib/zeppelin-0.6.0-bin-all/
[cloudera@quickstart zeppelin-0.6.0-bin-all]$ ls bin
common.cmd      functions.sh            interpreter.sh      zeppelin.sh
common.sh       install-interpreter.sh  zeppelin.cmd
functions.cmd   interpreter.cmd         zeppelin-daemon.sh
[cloudera@quickstart zeppelin-0.6.0-bin-all]$ bin/zeppelin-daemon.sh start
Zeppelin start                                             [  OK  ]
[cloudera@quickstart zeppelin-0.6.0-bin-all]$
```

Accessing Zeppelin UI

Open browser and use port 8080 to access Zeepelin notebook <quickstart.cloudera:8080>

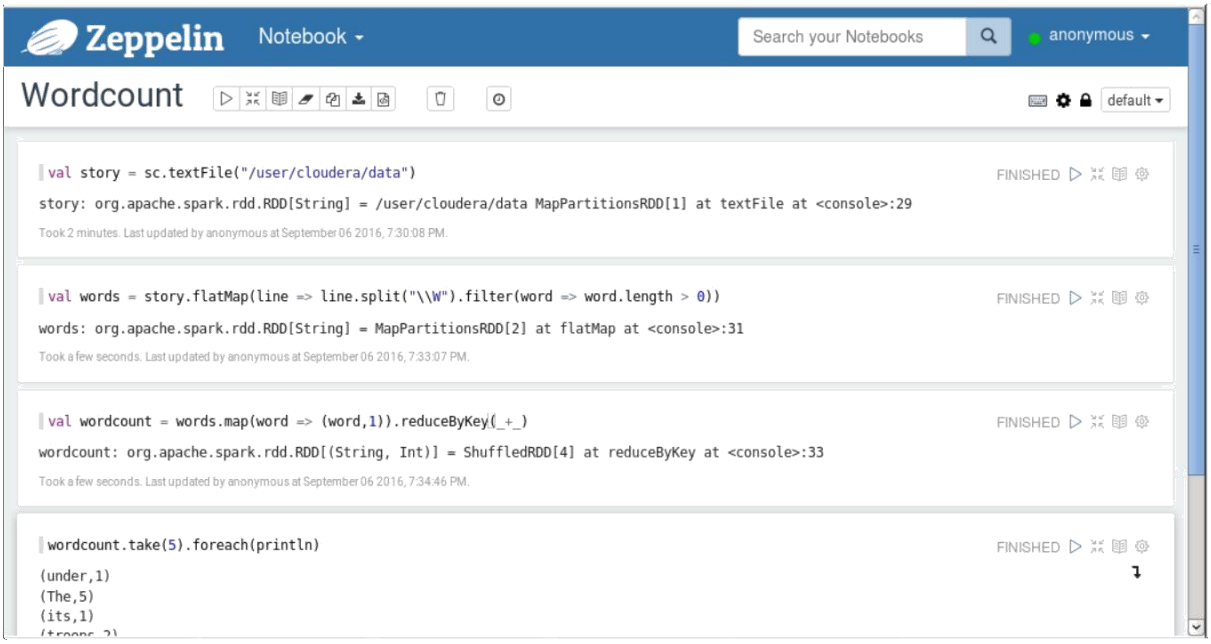


WordCount Example

Create a new note, name it as 'WordCount' and run following commands from zeepelin notebook.

```
val story = sc.textFile("/user/cloudera/data")

val words = story.flatMap(line => line.split("\\W").filter(word =>
word.lenghth > 0))

val wordcount = words.map(word => (word,1)).reduceByKey(_+_)
```

```
wordcount.take(5).foreach(println)
```



```
val department1 = new Department("123456", "Computer Science")
val department2 = new Department("789012", "Mechanical Engineering")
val department3 = new Department("345678", "Theater and Drama")
val department4 = new Department("901234", "Indoor Recreation")
```

Create the Employees

```
val employee1 = new Employee("michael", "armbrust",
"no- reply@berkeley.edu", 100000)
val employee2 = new Employee("xiangrui", "meng",
"no- reply@stanford.edu", 120000)
val employee3 = new Employee("matei", null, "no-
reply@waterloo.edu", 140000)
val employee4 = new Employee(null, "wendell",
"no- reply@princeton.edu", 160000)
```

Create the DepartmentWithEmployees instances from Departments and Employees

```
val departmentWithEmployees1 = new
DepartmentWithEmployees(department1, Seq(employee1, employee2))
```

```
val departmentWithEmployees2 = new
DepartmentWithEmployees(department2, Seq(employee3, employee4))
val departmentWithEmployees3 = new
DepartmentWithEmployees(department3, Seq(employee1, employee4))
val departmentWithEmployees4 = new
DepartmentWithEmployees(department4, Seq(employee2, employee3))
```

Create the first DataFrame from a List of the Case Classes.

```
val departmentsWithEmployeesSeq1 =
Seq(departmentWithEmployees1, departmentWithEmployees2)
val df1 = departmentsWithEmployeesSeq1.toDF()
df1.show
```

Create a 2nd DataFrame from a List of Case Classes.

```
val departmentsWithEmployeesSeq2 =
Seq(departmentWithEmployees3, departmentWithEmployees4)
val df2 = departmentsWithEmployeesSeq2.toDF()
df2.show
```

Union 2 DataFrames.

```
val unionDF = df1.unionAll(df2)
unionDF.show
```

Write the Unioned Dataframe to Parquet File

```
unionDF.write.parquet("/tmp/databricks-df-example.parquet")
```

Read a DataFrame from the Parquet file.

```
val parquetDF = sqlContext.read.parquet("/tmp/databricks-
df-example.parquet")
```

Exercise 4: Produce and Consume Apache Kafka Messages

Start the Zookeeper server

1. Open a new terminal window and browse to the location "/usr/lib/zookeeper" using cd. Start the zookeeper by command

```
bin/zkServer.sh start
```

```
[cloudera@quickstart ~]$ cd /usr/lib/zookeeper
[cloudera@quickstart zookeeper]$ bin/zkServer.sh start
JMX enabled by default
Using config: /usr/lib/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
[cloudera@quickstart zookeeper]$
```

2. You can now validate that Zookeeper is running correctly in standalone mode by connecting to the client port and sending the four letter command "srvr".

```
telnet localhost 2181
```

```
[cloudera@quickstart zookeeper]$ telnet localhost 2181
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
srvr
Zookeeper version: 3.4.5-cdh5.12.0--1, built on 06/29/2017 11:30 GMT
Latency min/avg/max: 0/0/1303
Received: 30491
Sent: 30604
Connections: 4
Outstanding: 0
Zxid: 0xe14
Mode: standalone
Node count: 737
Connection closed by foreign host.
[cloudera@quickstart zookeeper]$
```

Creating a Kafka Topic

1. Open a new terminal window and create a Kafka topic named `weblogs` that will contain messages representing lines in `Loudacre's web server logs`.
Since your exercise environment is a single-node cluster running on a virtual machine, use a replication factor of 1 and a single partition.

Navigate to the location where Kafka is installed, which is "/usr/lib/kafka".

```
$ bin/kafka-topics --create \ --zookeeper localhost:2181 \ --
replication-factor 1 \ --partitions 1 \
--topic weblogs
```

Here, Topic name
The name of the topic that you wish to create.

Replication Factor
The number of replicas of the topic to maintain within the cluster.

Partitions
The number of partitions to create for the topic.

```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic weblogs
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Created topic "weblogs".
```

1. Display all Kafka topics to confirm that the new topic you just created is listed:

```
$ bin/kafka-topics --list --zookeeper localhost:2181
```

```
[cloudera@quickstart kafka]$ kafka-topics --list --zookeeper localhost:2181
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
weblogs
[cloudera@quickstart kafka]$
```

2. Describe the Kafka topic created

```
$ bin/kafka-topics.sh --describe --zookeeper localhost:2181
--topic weblogs
```

```
[cloudera@quickstart kafka]$ kafka-topics.sh --describe --zookeeper localhost:2181 --topic weblogs
bash: kafka-topics.sh: command not found
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic weblogs
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Topic:weblogs   PartitionCount:1        ReplicationFactor:1     Configs:
        Topic: weblogs  Partition: 0    Leader: 0       Replicas: 0     Isr: 0
```

Producing and Consuming Messages

You will now use Kafka command line utilities to start producers and consumers for the topic created earlier.

3. Start a Kafka producer for the weblogs topic:

```
$ kafka-console-producer --broker-list localhost:9092 --topic
weblogs
```

Tip : This exercise involves using multiple terminal windows. To avoid confusion, set a different title for each one by selecting Set Title... on the Terminal menu:
Set the title for this window to "Kafka Producer."

4. Publish a test message to the weblogs topic by typing the message text and then pressing Enter. For example:

```
test weblog entry 1
```

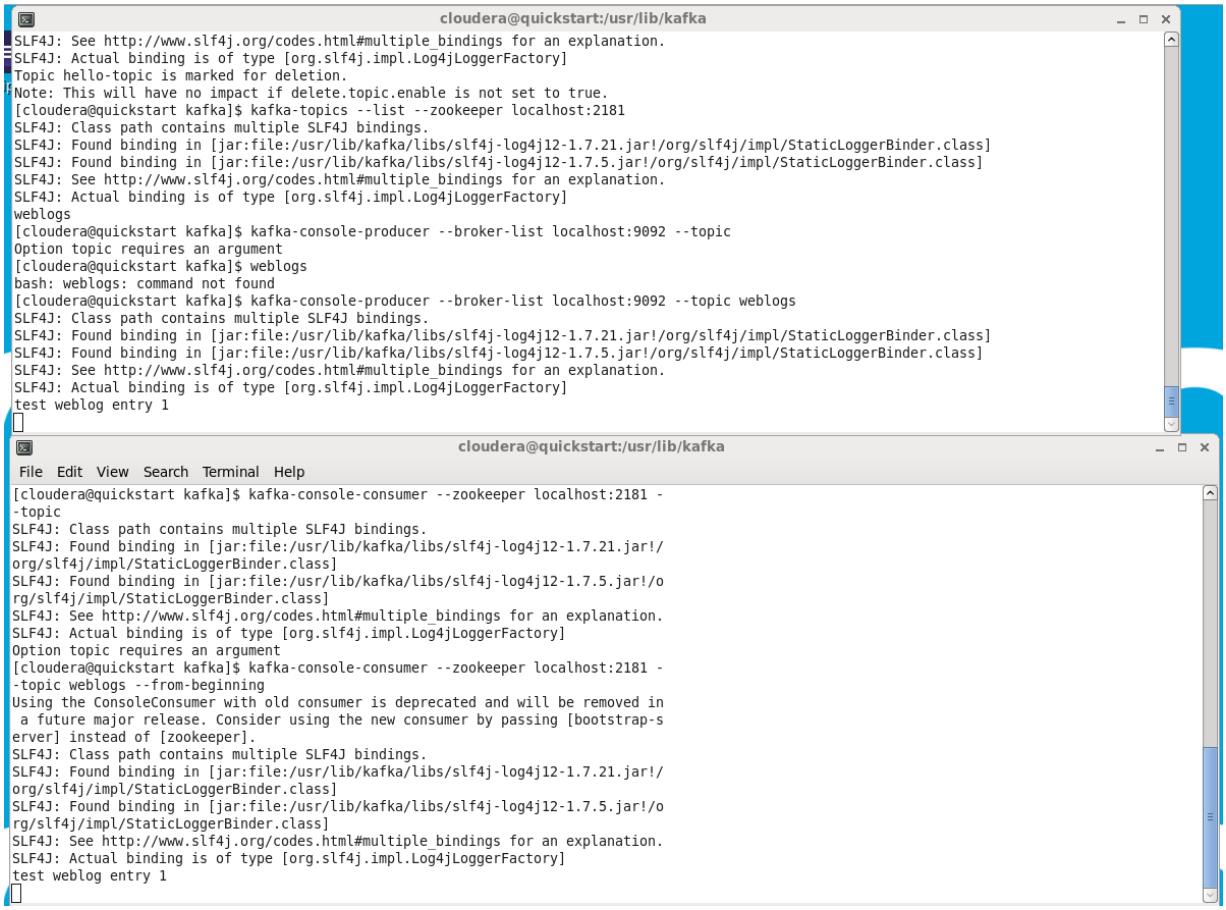
5. Open a new terminal window and adjust it to fit on the window beneath the producer window. Set the title for this window to "Kafka Consumer."

6. In the new terminal window, start a Kafka consumer that will read from the beginning of the weblogs topic:

```
$ kafka-console-consumer --zookeeper localhost:2181 --topic
weblogs --from-beginning
```

You should see the status message you sent using the producer displayed on the consumer's console, such as:

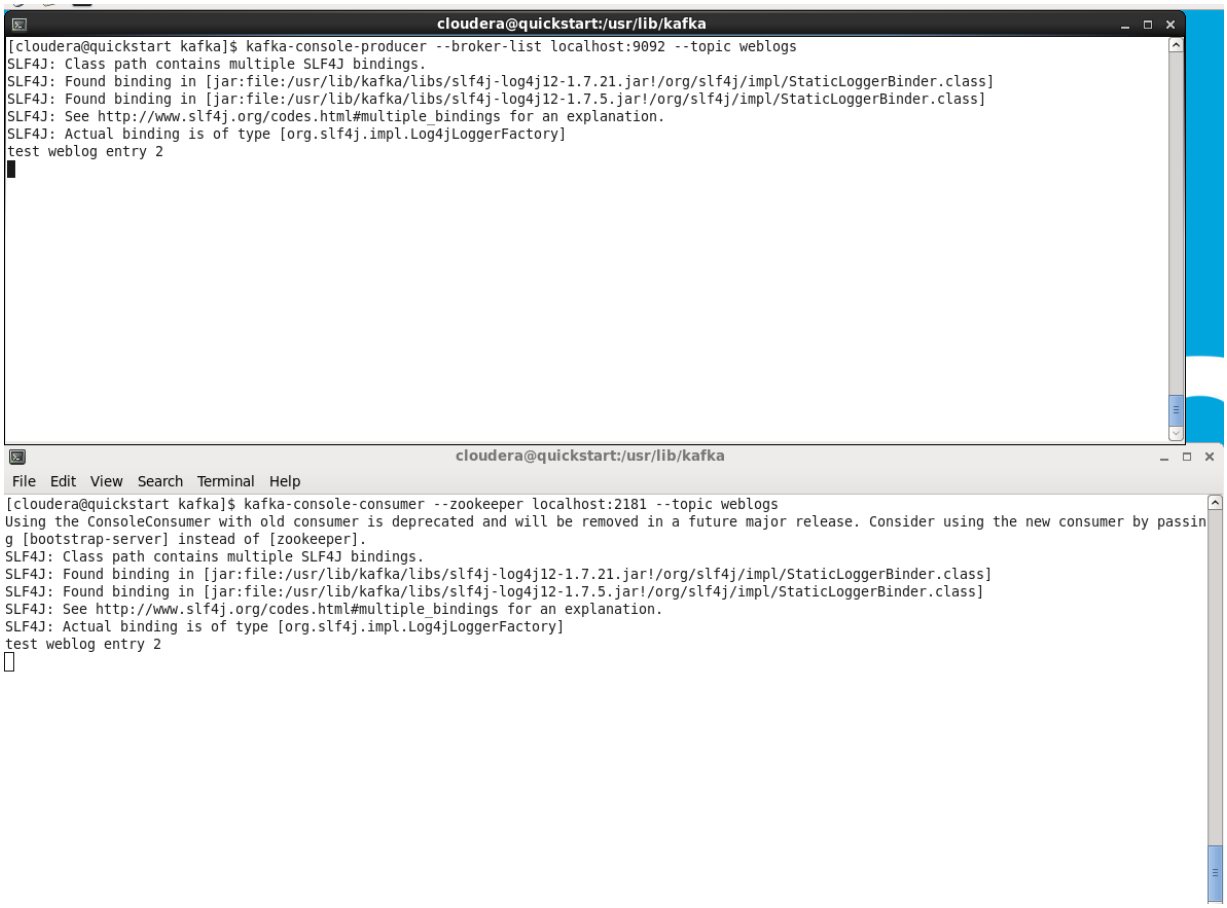
```
test weblog 1
```



7. Press Ctrl+C to stop the weblogs consumer, and restart it, but this time omit the from-beginning option to this command. You should see that no messages are displayed.

8. Switch back to the producer window and type another test message into the terminal, followed by the Enter key:

```
test weblog entry 2
```



9. Return to the consumer window and verify that it now displays the alert message you published from the producer in the previous step.

Cleaning Up

10. Press Ctrl+C in the consumer terminal window to end its process.

11. Press Ctrl+C in the producer terminal window to end its process.

Exercise 5: Alter Apache Kafka Topics

Let us try altering some parameters for the topic, but instead of using the earlier created topic <weblogs>, we will create another topic “hello-topic” to alter some parameters.

Exercise: Create a topic called hello-topic with replication factor 1 and partition 1.

1. We will add more partitions in the topic created as “hello-topic”. Below command will add 10 more partitions to the hello-topic topic. Note that before, the topic has only 1 partition.

```
bin/kafka-topics.sh --alter --zookeeper localhost:2181 --
partitions 11 --topic hello-topic
```

```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --alter --zookeeper localhost:2181 --partitions 11 --topic hello-topic
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
WARNING: If partitions are increased for a topic that has a key, the partition logic or ordering of the messages will be affected
Adding partitions succeeded!
[cloudera@quickstart kafka]$
```

Let us verify the number of partitions is being changed with the command

```
bin/kafka-topics.sh --describe --zookeeper localhost:2181 --
topic hello-topic
```

```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic hello-topic
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Topic:hello-topic       PartitionCount:11       ReplicationFactor:1     Configs:
        Topic: hello-topic      Partition: 0    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 1    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 2    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 3    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 4    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 5    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 6    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 7    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 8    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 9    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 10   Leader: 0       Replicas: 0     Isr: 0
[cloudera@quickstart kafka]$
```

Tip: REDUCING PARTITION COUNTS

It is not possible to reduce the number of partitions for a topic. The reason this is not supported is because deleting a partition from a topic would cause part of the data in that topic to be deleted as well, which would be inconsistent from a client point of view. In addition, trying to redistribute the data to remaining partitions would be difficult and result in out-of-order messages. Should you need to reduce the number of partitions, you will need to delete the topic and recreate it.

2. Add configurations to the Kafka topic

   The general syntax is:

```
bin/kafka-topics.sh --alter --zookeeper localhost:2181 --topic
kafkatopic --config <key>=<value>
```

There are various configuration fields we can set for the topic, here we will set up the max.message.bytes - this is the largest size of the message the broker will allow to be appended to the topic. This size is validated pre-compression. (Defaults to broker's message.max.bytes.)

```
bin/kafka-topics.sh --zookeeper localhost:2181 --alter --topic
hello-topic --config max.message.bytes=128000
```

```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --zookeeper localhost:2181 --alter --topic hello-topic --config max.message.bytes=128000
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
WARNING: Altering topic configuration from this script has been deprecated and may be removed in future releases.
         Going forward, please use kafka-configs.sh for this functionality
Updated config for topic "hello-topic".
```

3. To remove above overridden configuration, we can use command:

```
bin/kafka-topics.sh --zookeeper localhost:2181 --alter --topic
hello-topic --delete-config max.message.bytes
```

Exercise 6: Delete a topic

If a topic is no longer needed, it can be deleted in order to free up these resources. In order to perform this action, the brokers in the cluster must have been configured with the delete.topic.enable option set to true. If this option has been set to false, then the request to delete the topic will be ignored.

We will delete the topic created with the below command

```
bin/kafka-topics.sh --delete --zookeeper localhost:2181 --
topic hello-topic
```

```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --delete --zookeeper localhost:2181 --topic hello-topic
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Topic hello-topic is marked for deletion.
Note: This will have no impact if delete.topic.enable is not set to true.
```

Lets verify if the topic has been deleted, with the –list command

```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --zookeeper localhost:2181 --list
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
weblogs
[cloudera@quickstart kafka]$
```

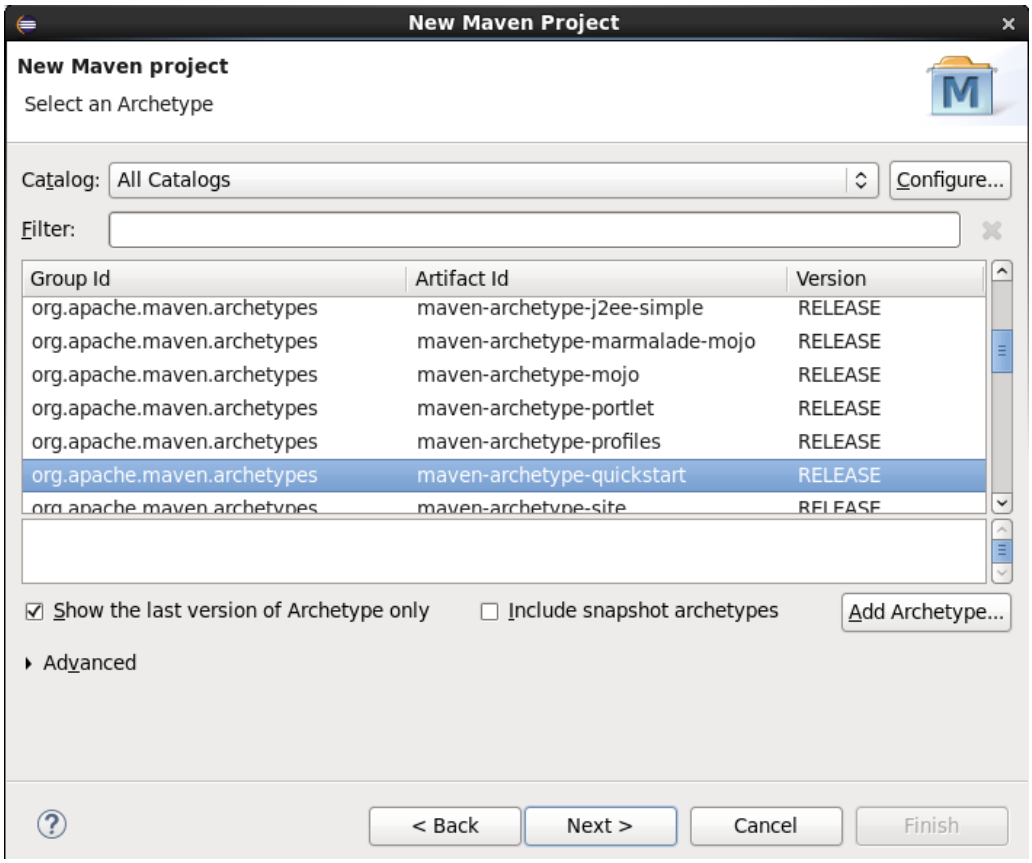
Tip: DATA LOSS AHEAD

Deleting a topic will also delete all its messages. This is not a reversible operation, so make sure it executed carefully.

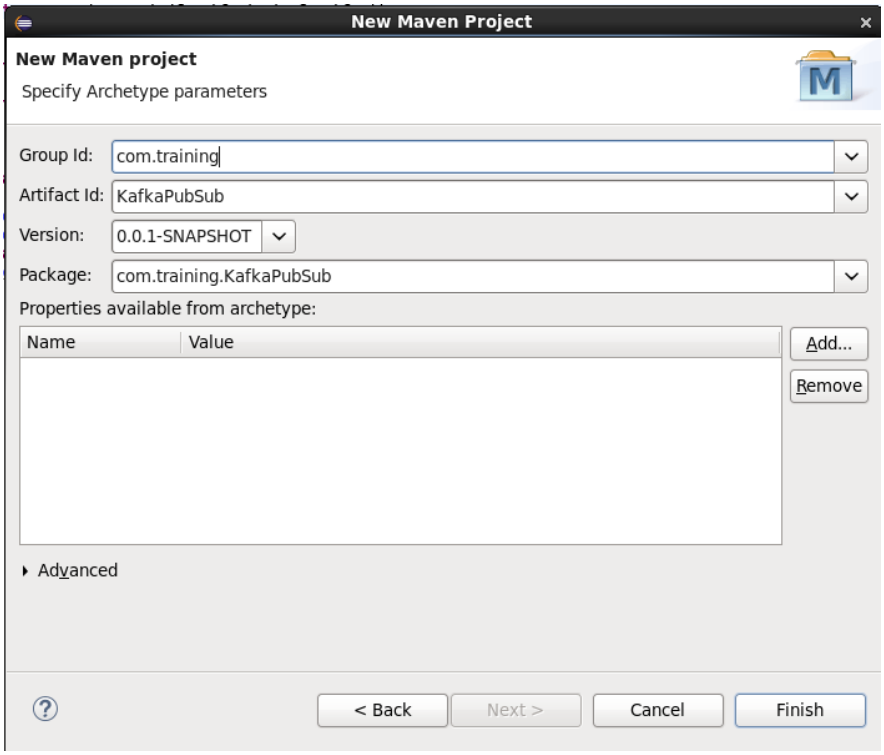
Exercise 7: Building Spark Application using Eclipse IDE Apache Kafka Java Client API

Goal: We are going to create a producer and consumer by using **Apache Kafka Java client API.**

1. Create new Maven Project using the menu bar option File –> Import –> Maven –> New Maven Projects Project. Give a name of your choice for the project.



Click on Next.

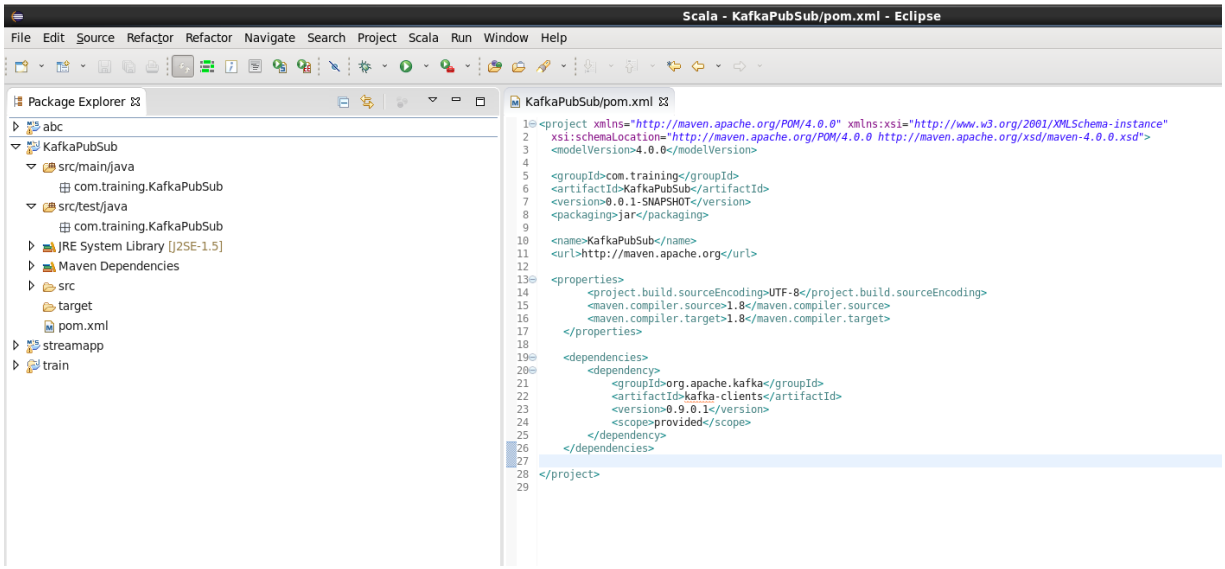


Click on Finish.

2. Update the maven pom.xml with the following dependences. Keep other content of
pom xml as is. Remember to add the below content right under the xml root called <project>.

```
<properties>
		<project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
		<maven.compiler.source>1.7</maven.compiler.source>
		<maven.compiler.target>1.7</maven.compiler.target>
	</properties>

	<dependencies>
		<dependency>
			<groupId>org.apache.kafka</groupId>
			<artifactId>kafka-clients</artifactId>
			<version>0.9.0.1</version>
			<scope>provided</scope>
		</dependency>
	<dependency>
	<groupId>org.apache.kafka</groupId>
	<artifactId>kafka_2.11</artifactId>
	<version>0.9.0.1</version>
	</dependency>
	</dependencies>
```

We use the Kafka-clients-0.9.0.1 library for this example. The Java compiler 1.7.



3. Now add a new Java class file to project, Right click on project > New > Class > ProducerTest.

```
public class ProducerTest {

 public static void main(String[] args) {
  Properties props = new Properties();
  props.put("bootstrap.servers", "localhost:9092");
  props.put("acks", "all");
  props.put("retries", 0);
  props.put("batch.size", 16384);
  props.put("linger.ms", 1);
  props.put("buffer.memory", 33554432);
  props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
  props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
  Producer<String, String> producer = null;
  try {
   producer = new KafkaProducer<>(props);
   for (int i = 0; i < 100; i++) {
    String msg = "Message " + i;
    producer.send(new ProducerRecord<String, String>("HelloKafkaTopic", msg));
    System.out.println("Sent:" + msg);
   }
  } catch (Exception e) {
   e.printStackTrace();

  } finally {
   producer.close();
  }

 }

}
```

A Kafka producer has three mandatory properties:

bootstrap.servers
List of host:port pairs of brokers that the producer will use to establish initial connection to the Kafka cluster. This list doesn't need to include all brokers, since the producer will get more information after the initial connection. But it is

recommended to include at least two, so in case one broker goes down, the producer will still be able to connect to the cluster.

key.serializer
Name of a class that will be used to serialize the keys of the records we will produce to Kafka. Kafka brokers expect byte arrays as keys and values of messages. Tthe producer has to know how to convert these objects to byte arrays. key.serializer should be set to a name of a class that implements the org.apache.kafka.common.serialization.Serializer interface. The producer will use this class to serialize the key object to a byte array.
The Kafka client package includes
ByteArraySerializer (which doesn't do much),
StringSerializer, and
IntegerSerializer, so if you use common types, there is no need to implement your own serializers.
Setting key.serializer is required even if you intend to send only values.

value.serializer
Name of a class that will be used to serialize the values of the records we will produce to Kafka. The same way you set key.serializer to a name of a class that will serialize the message key object to a byte array, you set value.serializer to a class that will serialize the message value object.

4. Now let us add a new Java class file to project, Right click on project >New > Class > ConsumerTest

```
public class ConsumerTest {

 public static void main(String[] args) {
  Properties props = new Properties();
  props.put("bootstrap.servers", "localhost:9092");
  props.put("group.id", "group-1");
  props.put("enable.auto.commit", "true");
  props.put("auto.commit.interval.ms", "1000");
  props.put("auto.offset.reset", "earliest");
  props.put("session.timeout.ms", "30000");
  props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
  props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

  KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<>(props);
  kafkaConsumer.subscribe(Arrays.asList("HelloKafkaTopic"));
  while (true) {
   ConsumerRecords<String, String> records = kafkaConsumer.poll(100);
   for (ConsumerRecord<String, String> record : records) {
    System.out.println("Partition: " + record.partition() + " Offset: " + record.offset()
      + " Value: " + record.value() + " ThreadID: " + Thread.currentThread().getId());
   }
  }

 }

}
```

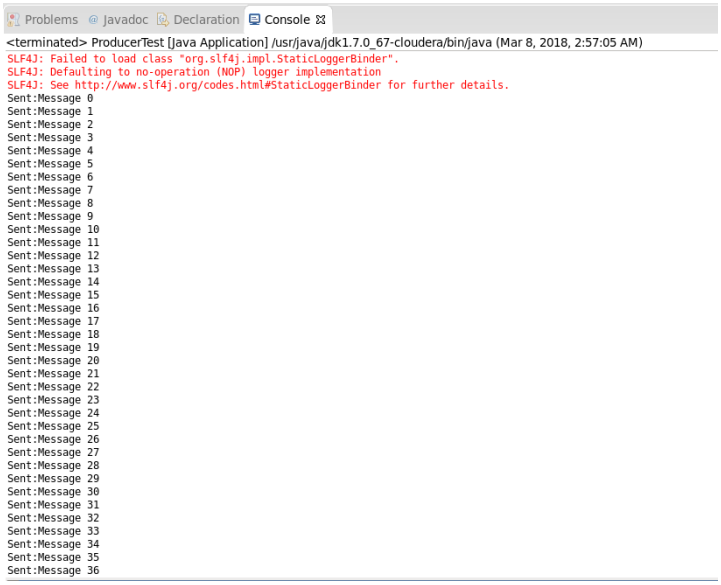
5. Now we will test our ProducerTest sending some messages, which will be received by ReceiverTest, before testing ensure that

   Topic 'mytopic' is created.

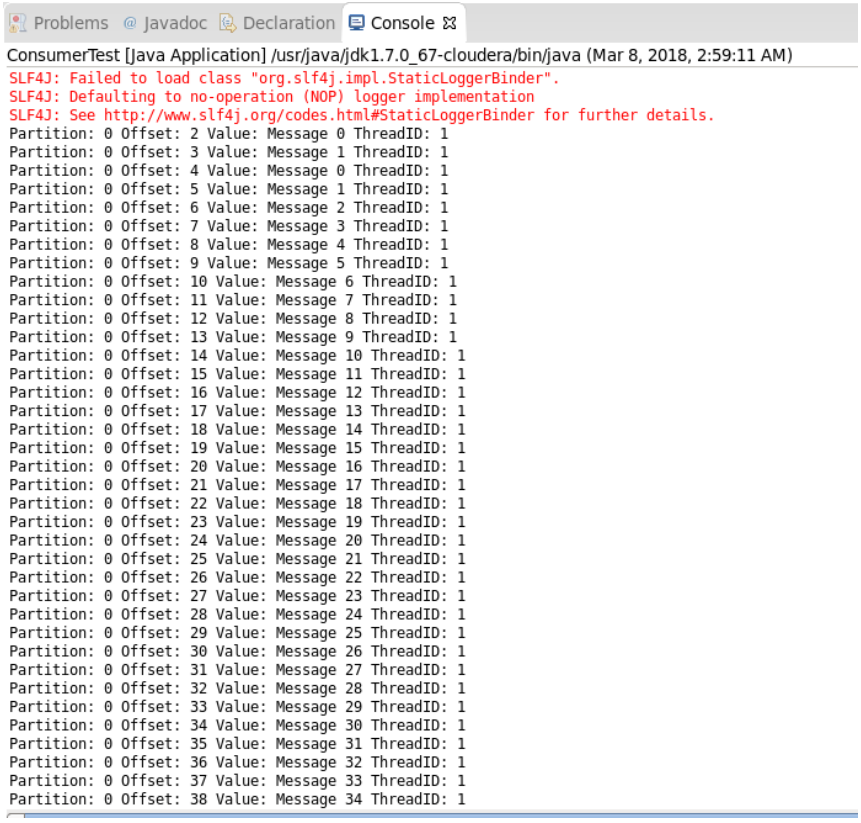
   Zookeeper should be running on localhost:2181.

   Kafka should be running on localhost:9092.

   Run ProducerTest.java > Java Application, The console will print the messages



   Run ConsumerTest.java > Java Application, the console of the receiver will receive all the messages from the broker.



Terminate the ConsumerTest.java class.

Assignment: In this assignment, we will use Sending a Message Synchronously

1. Create a Producer class as “KafkaPublisher” for the topic “mytopic”.
2. Imports
   import org.apache.kafka.clients.consumer.ConsumerRecords;
   import org.apache.kafka.clients.consumer.ConsumerRecord;
   import org.apache.kafka.clients.producer.ProducerRecord;
3. Set the mandatory values for the KafkaPublisher, which is the producer class
   bootstrap.servers=broker2:9092
   key.serializer=org.apache.kafka.common.serialization.StringSerializer
   value.serializer=org.apache.kafka.common.serialization.StringSerializer
4. Set the values in a Properties object.
5. Instead of Fire-and-forget which was done before we will use the synchronous way of sending the message , let us send a message, the send() method returns a Future object, and we use get() to wait on the future and see if the send() was successful or not.
6. The message record is created as
   ProducerRecord<String, String> record =
   new ProducerRecord<>("mytopic", "Precision Products", "France");
   The ProducerRecord objects we created included a topic name, key, and value.
7. Send the message i.e. record using producer.send(record).get() in the KafkaPublisher.java class.
8. Create a Consumer class as KafkaSubscriber.
9. Set the three mandatory properties: bootstrap.servers, key.deserializer, and value.deserializer (deserializer should be of same type as of serializer).
10. Now set the fourth property, which is not strictly mandatory, but for now we will pretend it is. The property is group.id and it specifies the consumer group the KafkaSubscriber instance belongs to.

    ```
    props.put("group.id", "testgroup")
    ```

11. Poll the message on the topic mytopic using the poll() method

    ```
    ConsumerRecords<String, String> records = consumer.poll(100);
      for (ConsumerRecord<String, String> record : records)
      {
        System.out.printf("topic = %s, partition = %s, offset =
          %d, customer = %s, country = %s\n",
            record.topic(), record.partition(),
              record.offset(), record.key(), record.value()); 1
      }
    ```

12. Run the KafkaPublisher as Java Application and also run the KafkaSubscriber as Java Application.
13. The received message will be printed in the console.

Exercise 8: IMPLEMENTING A CUSTOM PARTITIONING STRATEGY

We can create a class implementing the org.apache.kafka.clients.producer.Partitioner interface. This custom Partitioner will implement the business logic to decide where messages are sent.

Use Case: We have a retail site that consumers can use to order products anywhere in the world. Based on usage, we know that most consumers are in either the United States or India. We want to partition our application to send orders from the US or India to their own respective consumers, while orders from anywhere else will go to a third consumer.

Let us create a CountryPartitioner that implements the org.apache.kafka.clients.producer.Partitioner interface as java class.

1. Create a maven project and define all the dependencies.
2. Create a Java class as CountryPartitioner

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.kafka.clients.producer.Partitioner;
import org.apache.kafka.common.Cluster;

public class CountryPartitioner implements Partitioner {
	private static Map<String, Integer> countryToPartitionMap;

	public void configure(Map<String, ?> configs) {
		System.out.println("Inside CountryPartitioner.configure " + configs);
		countryToPartitionMap = new HashMap<String, Integer>();
		for (Map.Entry<String, ?> entry : configs.entrySet()) {
			if (entry.getKey().startsWith("partitions.")) {
				String keyName = entry.getKey();
				String value = (String) entry.getValue();
				System.out.println(keyName.substring(11));
				int paritionId = Integer.parseInt(keyName.substring(11));
				countryToPartitionMap.put(value, paritionId);
			}
		}
	}

	public int partition(String topic, Object key, byte[] keyBytes,
			Object value, byte[] valueBytes, Cluster cluster) {
		List partitions = cluster.availablePartitionsForTopic(topic);
```

```
                String valueStr = (String) value;
                String countryName = ((String) value).split(":")[0];
                if (countryToPartitionMap.containsKey(countryName)) {
                        // If the country is mapped to particular partition return it
                        return countryToPartitionMap.get(countryName);
                } else {
                        // If no country is mapped to particular partition distribute
                        // between remaining partitions
                        int noOfPartitions = cluster.topics().size();
                        return value.hashCode() % noOfPartitions
                                        + countryToPartitionMap.size();
                }
        }

        public void close() {
        }
}
```

When we create a custom partitioner, We must implement the following methods:

- configure() - This method initializes functions specific to the application's business logic, such as connecting to a database. In this case we want a fairly generic partitioner that takes countryName as a property. We can then use configProperties.put("partitions.0","USA") to map the flow of messages to partitions.
- partition() - The Producer API calls partition() once for every message. In this case we'll use it to read the message and parse the name of the country from the message. If the name of the country is in the countryToPartitionMap, it will return partitionId stored in the Map. If not, it will hash the value of the country and use it to calculate which partition it should go to.
- close()- We call close() to shut down the partitioner.

Note: We set a config property with a key equal to the value of ProducerConfig.PARTITIONER_CLASS_CONFIG, which matches the fully qualified name of our CountryPartitioner class. We also set countryName to partitionId, thus mapping the properties that we want to pass to CountryPartitioner.

3. Now let us create the Producer class

```
import java.util.Properties;
import java.util.Scanner;

import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;

public class Producer {
```

```
   private static Scanner in;
   @SuppressWarnings("unchecked")
         public static void main(String[] argv)throws Exception {

      String topicName = "part-demo"/*argv[0]*/;
      in = new Scanner(System.in);
      System.out.println("Enter message(type exit to quit)");

      //Configure the Producer
      Properties configProperties = new Properties();
      configProperties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,"localhost:9092");

configProperties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,"org.apache.kafka.co
mmon.serialization.ByteArraySerializer");

configProperties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,"org.apache.kafka.
common.serialization.StringSerializer");


configProperties.put(ProducerConfig.PARTITIONER_CLASS_CONFIG,CountryPartitioner.class
.getCanonicalName());
      configProperties.put("partitions.0","USA");
      configProperties.put("partitions.1","India");

      org.apache.kafka.clients.producer.Producer producer = new
KafkaProducer(configProperties);
      String line = in.nextLine();
      while(!line.equals("exit")) {
         ProducerRecord<String, String> rec = new ProducerRecord<String,
String>(topicName, line);

         producer.send(rec, new Callback() {
            public void onCompletion(RecordMetadata metadata, Exception exception) {
               System.out.println("Message sent to topic ->" + metadata.topic()+ " ,parition->"
+ metadata.partition() +" stored at offset->" + metadata.offset());
            }
         });
         line = in.nextLine();
      }
      in.close();
      producer.close();
   }
}
```

4. Let us create partitioned consumer

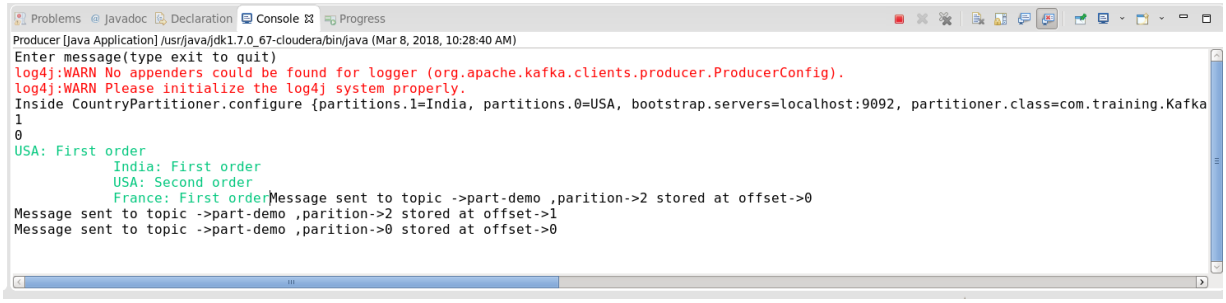
```
import java.util.Arrays;
import java.util.Collection;
import java.util.Properties;
import java.util.Scanner;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRebalanceListener;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.TopicPartition;
import org.apache.kafka.common.errors.WakeupException;

public class Consumer {
   private static Scanner in;
   private static boolean stop = false;

   public static void main(String[] argv) throws Exception {
      in = new Scanner(System.in);
      String topicName = "part-demo"/*argv[0]*/;
     // String groupId = argv[1];

      ConsumerThread consumerThread = new ConsumerThread(topicName, "group1");
      consumerThread.start();
      String line = "";
      while (!line.equals("exit")) {
         line = in.next();
      }
      consumerThread.getKafkaConsumer().wakeup();
      System.out.println("Stopping consumer .....");
      consumerThread.join();
   }

   private static class ConsumerThread extends Thread {
      private String topicName;
      private String groupId;
      private KafkaConsumer<String, String> kafkaConsumer;

      public ConsumerThread(String topicName, String groupId) {
         this.topicName = topicName;
         this.groupId = groupId;
      }

      public void run() {
```

```
        Properties configProperties = new Properties();
        configProperties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
"localhost:9092");
        configProperties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
"org.apache.kafka.common.serialization.StringDeserializer");
        configProperties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
"org.apache.kafka.common.serialization.StringDeserializer");
        configProperties.put(ConsumerConfig.GROUP_ID_CONFIG, groupId);

        //Figure out where to start processing messages from
        kafkaConsumer = new KafkaConsumer<String, String>(configProperties);
        kafkaConsumer.subscribe(Arrays.asList(topicName), new
ConsumerRebalanceListener() {
          public void onPartitionsRevoked(Collection<TopicPartition> partitions) {
            System.out.printf("%s topic-partitions are revoked from this consumer\n",
Arrays.toString(partitions.toArray()));
          }
          public void onPartitionsAssigned(Collection<TopicPartition> partitions) {
            System.out.printf("%s topic-partitions are assigned to this consumer\n",
Arrays.toString(partitions.toArray()));
          }
        });
        //Start processing messages
        try {
          while (true) {
            ConsumerRecords<String, String> records = kafkaConsumer.poll(100);
            for (ConsumerRecord<String, String> record : records)
              System.out.println(record.value());
          }
        } catch (WakeupException ex) {
          System.out.println("Exception caught " + ex.getMessage());
        } finally {
          kafkaConsumer.close();
          System.out.println("After closing KafkaConsumer");
        }
      }

      public KafkaConsumer<String, String> getKafkaConsumer() {
        return this.kafkaConsumer;
      }
    }
}
```

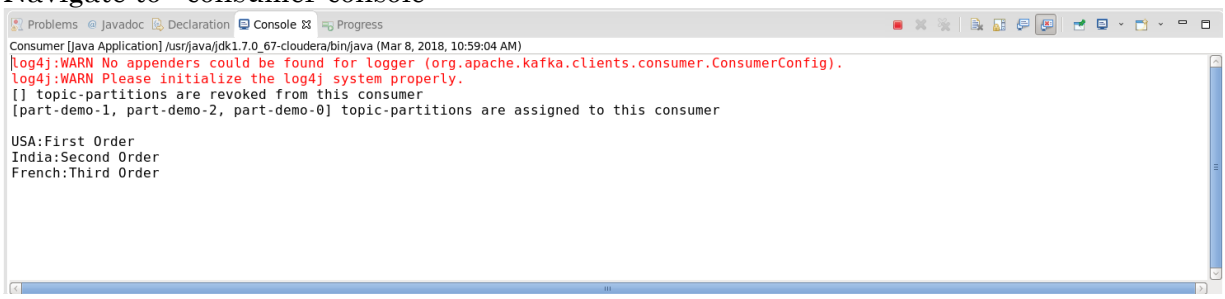
5. Let us test the application by sending some messages to the topic “demo-part” with 3 partitions. Create a topic named part-demo with three partitions and one replication factor:

```
bin/kafka-topics.sh --create --zookeeper
localhost:2181 --replication-factor 1 --partitions
3 --topic part-demo
```

6. Start a producer as JavaApplication.
7. Start the consumer as JavaApplication.
8. Type some messages into your producer console and verify whether the messages are routed to the correct consumer:
   Navigate to <Producer-console>



   Navigate to <consumer-console>



Exercise 9: How to write a custom serializer in Apache Kafka

Apache Kafka supports some kinds of serializer/deserializer as below:

- ByteArray
- Integer
- Long
- String

If there is not enough for your purpose. You can try to implement your own ones.

Use Case: We have User created from the producer which is being sent to the Kafka Server and is being received by the consumer. Kafka depends on Serializer and Deserializer so that Producer and Consumer both know how to communicate and understand the messages and both KafkaProducer and KafkaConsumer needs key and value serializer. In Case of custom objects, we will create custom serializer in Kafka.

1. Create a Maven Project with required artifactId and groupId(as per your choice).
2. Update Maven Pom.xml

```
<properties>
```

```
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.7</maven.compiler.source>
        <maven.compiler.target>1.7</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>0.9.0.1</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.7.4</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
```

3. We will create the serializer and deserializer for User Objects, the below Java classes will be created

   User. java. Encapsulates user information. Producer will create and send users to broker while consumer will receive the objects from the broker.

   UserSerializer.java. Convert the User object to byte array

   UserDeserializer.java. Convert the byte array to User object

   UserProducerThread.java. The producer class

   UserConsumerThread.java. The consumer class

   KakfaCustomSerializerMain.java. The entry point, used for testing purpose.

4. User.java for encapsulating all the user attributes

```
public class User {
        private Long id;
        private String userName;
        private String firstName;
        private String lastName;
        private int age;

        public User() {

        }

        public User(Long id, String userName, String firstName, String lastName, int age) {
         super();
```

```
        this.id = id;
        this.userName = userName;
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
      }

      /*
       * (non-Javadoc)
       *
       * @see java.lang.Object#toString()
       */
      @Override
      public String toString() {
        return "User [id=" + id + ", userName=" + userName + ", firstName=" + firstName +
", lastName="
            + lastName + ", age=" + age + "]";
      }
      //getters and setters

    }
```

Note: Generate the getters and setters for the User attributes.

5. Now we will create the UserSerializer.java class

```
import java.util.Map;
import org.apache.kafka.common.serialization.Serializer;
import com.fasterxml.jackson.databind.ObjectMapper;

public class UserSerializer implements Serializer<User> {

    @Override
    public void close() {

    }

    @Override
    public void configure(Map<String, ?> arg0, boolean arg1) {

    }

    @Override
    public byte[] serialize(String arg0, User arg1) {
        byte[] retVal = null;
        ObjectMapper objectMapper = new ObjectMapper();
        try {
```

```
                retVal = objectMapper.writeValueAsString(arg1).getBytes();
            } catch (Exception e) {
                e.printStackTrace();
            }
            return retVal;
        }

}
```

Note: We have to implement the Serializer interface of Apache Kafka client API. The main method we need to implement is the serialize method. It simply converts the object into byte array and returns. Here, we utilize the ObjectMapper object from the jackson-databind library to quickly serialize the whole object into byte array.

6. Next we will create the deserializer Java class as UserDeserializer, We need to implement the method deserialize(). And in here, we also utilize the ObjectMaper object to convert the byte array back to the User object

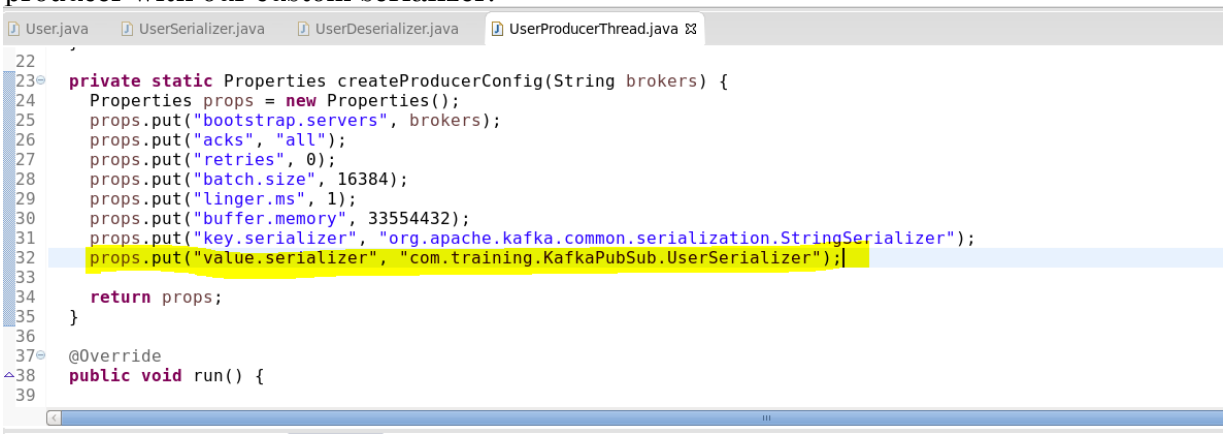
```
import java.util.Map;
import org.apache.kafka.common.serialization.Deserializer;
import com.fasterxml.jackson.databind.ObjectMapper;

public class UserDeserializer implements Deserializer<User> {

        @Override
        public void close() {

        }

        @Override
        public void configure(Map<String, ?> arg0, boolean arg1) {

        }

        @Override
        public User deserialize(String arg0, byte[] arg1) {
            ObjectMapper mapper = new ObjectMapper();
            User user = null;
            try {
                user = mapper.readValue(arg1, User.class);
            } catch (Exception e) {

                e.printStackTrace();
            }
            return user;
        }

}
```

7. Create a Producer thread as java class UserProducerThread.java. When run, it will send a list of 2 users to the broker.
   Note: Here is we registered the value.serializer property of the producer with our custom serializer.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;

public class UserProducerThread implements Runnable {

 private final KafkaProducer<String, User> producer;
 private final String topic;

 public UserProducerThread(String brokers, String topic) {
  Properties prop = createProducerConfig(brokers);
  this.producer = new KafkaProducer<String, User>(prop);
  this.topic = topic;
 }

 private static Properties createProducerConfig(String brokers) {
  Properties props = new Properties();
  props.put("bootstrap.servers", brokers);
  props.put("acks", "all");
  props.put("retries", 0);
  props.put("batch.size", 16384);
  props.put("linger.ms", 1);
  props.put("buffer.memory", 33554432);
  props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
  props.put("value.serializer", "com.howtoprogram.kafka.customserializer.UserSerializer");

  return props;
 }

 @Override
 public void run() {

  List<User> users = new ArrayList<>();
  users.add(new User(1l, "tom", "Tom", "Riddle", 40));
  users.add(new User(2l, "harry", "Harry", "Potter", 10));
  for (final User user : users) {
```

```
    producer.send(new ProducerRecord<String, User>(topic, user.getUserName(), user),
        new Callback() {
          public void onCompletion(RecordMetadata metadata, Exception e) {
            if (e != null) {
              e.printStackTrace();
            }
            System.out.println("Sent:" + user.toString());
          }
        });
    try {
      Thread.sleep(100);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }

  }

  // closes producer
  producer.close();

 }
}
```

Note: Important note here is we registered the value.serializer property of the producer with our custom serializer.

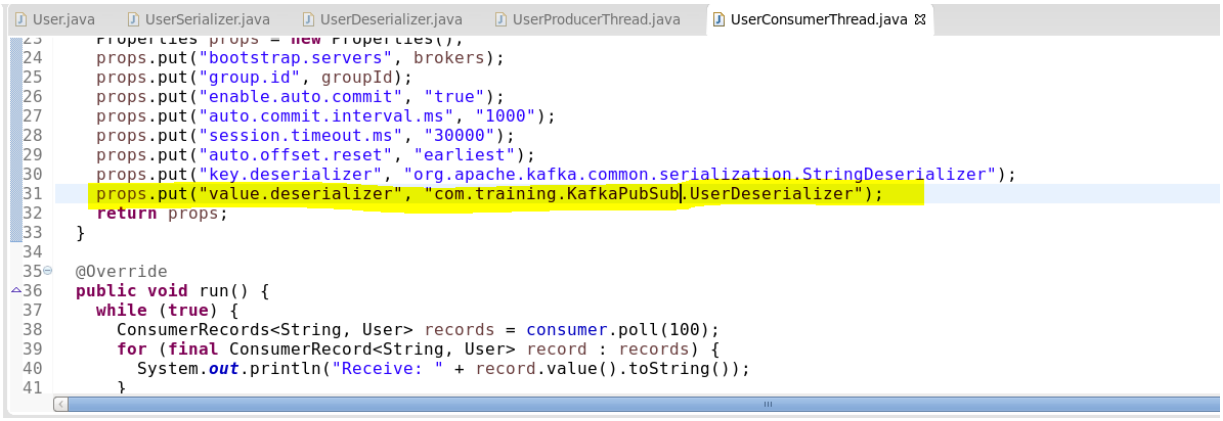


8. Create a Consumer class as UserConsumerThread.java it will poll the broker for the messages and print out to the console.

```
import java.util.Arrays;
import java.util.Properties;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
```

```
public class UserConsumerThread implements Runnable {

 private final KafkaConsumer<String, User> consumer;
 private final String topic;

 public UserConsumerThread(String brokers, String groupId, String topic) {
  Properties prop = createConsumerConfig(brokers, groupId);
  this.consumer = new KafkaConsumer<>(prop);
  this.topic = topic;
  this.consumer.subscribe(Arrays.asList(this.topic));
 }

 private static Properties createConsumerConfig(String brokers, String groupId) {
  Properties props = new Properties();
  props.put("bootstrap.servers", brokers);
  props.put("group.id", groupId);
  props.put("enable.auto.commit", "true");
  props.put("auto.commit.interval.ms", "1000");
  props.put("session.timeout.ms", "30000");
  props.put("auto.offset.reset", "earliest");
  props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
  props.put("value.deserializer", " com.training.KafkaPubSub.UserDeserializer");
  return props;
 }

 @Override
 public void run() {
  while (true) {
   ConsumerRecords<String, User> records = consumer.poll(100);
   for (final ConsumerRecord<String, User> record : records) {
    System.out.println("Receive: " + record.value().toString());
   }
  }
 }

}
```

Note : that we already registered the value.deserializer property of the consumer with our custom deserializer.



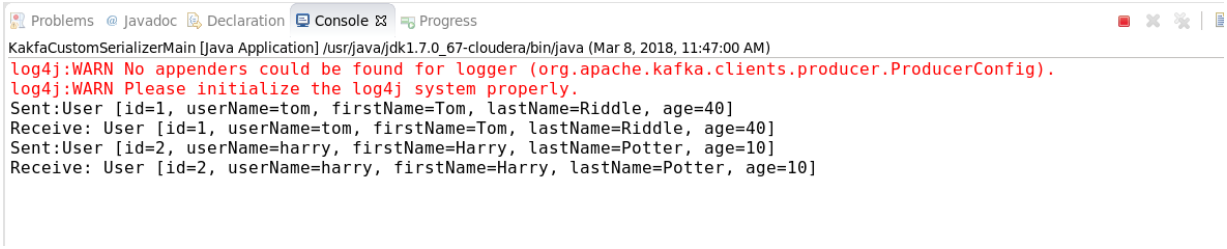
9. Test the application using the Custom serializer we just created using a test class. This will send the message to the broker using ProducerThread created by us before the user object will be serialized using the custom serializer. The message will be deserialized from the Kafka broker and polled by the ConsumerThread.

```
public final class KakfaCustomSerializerMain {

        public static void main(String[] args) {

          String brokers = "localhost:9092";
          String groupId = "group01";
          String topic = "UserMessageTopic10";

          if (args != null && args.length == 3) {
            brokers = args[0];
            groupId = args[1];
            topic = args[2];
          }

          // Start User Producer Thread
          UserProducerThread producerThread = new UserProducerThread(brokers, topic);
          Thread t1 = new Thread(producerThread);
          t1.start();

          // Start group of User Consumer Thread
          UserConsumerThread consumerThread = new UserConsumerThread(brokers,
groupId, topic);
          Thread t2 = new Thread(consumerThread);
          t2.start();

          try {
            Thread.sleep(100000);
          } catch (InterruptedException ie) {

          }
        }      }
```

10. Create the topic as UserMessageTopic10.

```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic UserMessageTopic10
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Created topic "UserMessageTopic10".
```

11. Open the KakfaCustomSerializerMain.java on the eclipse. Right click –> Run As –> Java Application or use the shortcut: Alt+Shift+x, j to start the main method.



Exercise 10: Collect Web Server Logs with Apache Flume

Files and Data Used in This Exercise

Exercise directory: `/home/cloudera/training/devsh/exercises/flume`

Data files (local): `/home/cloudera/training/data/weblogs/*`

In this exercise, you will run a Flume agent to ingest web log data from a local directory to HDFS.

Apache web server logs are generally stored in files on the local machines running the server. In this exercise, you will simulate an Apache server by placing provided web log files into a local spool directory, and then use Flume to collect the data.

Both the local and HDFS directories must exist before using the spooling directory source.

Creating an HDFS Directory for Flume-Ingested Data

1. Create a directory in HDFS called `/loudacre/weblogs_flume` to hold the data files Flume ingests:

```
$ hdfs dfs -mkdir -p /loudacre/weblogs_flume
```

```
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$ hdfs dfs -mkdir -p /loudacre/weblogs_flume
[cloudera@quickstart ~]$
```

Creating a Local Directory for Web Server Log Output

2. Create the spool directory into which the web log simulator will store data files for Flume to ingest. On the local Linux filesystem create `/flume/weblogs_spooldir`:

```
$ sudo mkdir -p /flume/weblogs_spooldir
```

3. Give all users the permissions to write to the `/flume/weblogs_spooldir`

directory:

```
$ sudo chmod a+w -R /flume
```

```
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$ sudo mkdir -p /flume/weblogs_spooldir
[cloudera@quickstart ~]$ sudo chmod a+w -R /flume
[cloudera@quickstart ~]$
```

Configuring Flume

A Flume agent configuration file has been provided for you:
`/home/cloudera/training/devsh/exercises/flume/spooldir.conf .`

Review the configuration file. You do not need to edit this file. Take note in particular of the following:

The source is a spooling directory source that pulls from the local `/flume/weblogs_spooldir` directory.

The sink is an HDFS sink that writes files to the HDFS `/loudacre/weblogs_flume` directory.

The channel is a memory channel.

```
[cloudera@quickstart flume]$ pwd
/home/cloudera/training/devsh/exercises/flume
[cloudera@quickstart flume]$ ls -ltr
total 8
-rw-r--r-- 1 cloudera cloudera 943 Nov 14  2016 spooldir.conf
-rwxr-xr-x 1 cloudera cloudera 841 Nov 14  2016 copy-move-weblogs.sh
[cloudera@quickstart flume]$
```

training

Running the Flume Agent

Next, start the Flume agent and copy the files to the spooling directory.

4. Change directories to the exercise directory.

```
$ cd /home/cloudera/training/devsh/exercises/flume
```

5. Start the Flume agent using the configuration you just reviewed:

```
$ flume-ng agent --conf /etc/flume-ng/conf \ --conf-file
spooldir.conf \ --name agent1 -Dflume.root.logger=INFO,console
```

```
^C[cloudera@quickstart flume]$ flume-ng agent --conf /etc/flume-ng/conf --conf-file spooldir.conf --name agent1 -D flume.root.logger=IN
,console
Info: Sourcing environment configuration script /etc/flume-ng/conf/flume-env.sh
Info: Including Hadoop libraries found via (/usr/bin/hadoop) for HDFS access
Info: Including HBASE libraries found via (/usr/bin/hbase) for HBASE access
Info: Including Hive libraries found via () for Hive access
```

6. Wait a few moments for the Flume agent to start up. You will see a message like in the logs: `Component type: SOURCE, name: webserver-log-source started`

```
09 Sep 2017 04:39:18,916 INFO  [lifecycleSupervisor-1-2] (org.apache.flume.instrumentation.MonitoredCounterGroup.register:119)  - Monit
ored counter group for type: SOURCE, name: webserver-log-source: Successfully registered new MBean.
09 Sep 2017 04:39:18,916 INFO  [lifecycleSupervisor-1-2] (org.apache.flume.instrumentation.MonitoredCounterGroup.start:95)  - Component
 type: SOURCE, name: webserver-log-source started
09 Sep 2017 04:40:31,766 INFO  [SinkRunner-PollingRunner-DefaultSinkProcessor] (org.apache.flume.sink.hdfs.HDFSDataStream.configure:57)
  - Serializer = TEXT, UseRawLocalFileSystem = false
```

Simulating Apache Web Server Output

7. Open a separate terminal window. Run the script to place the web log files in the `/flume/weblogs_spooldir` directory:

```
$ /home/cloudera/training/devsh/exercises/flume/copy-move-
weblogs.sh \ /flume/weblogs_spooldir
```

This script will create a temporary copy of the web log files and move them to the `spooldir` directory.

```
[cloudera@quickstart flume]$ ./copy-move-weblogs.sh /flume/weblogs_spooldir/
Copying and moving files to /flume/weblogs_spooldir/
[cloudera@quickstart flume]$
```

8. Return to the terminal that is running the Flume agent and watch the logging output. The output will give information about the files Flume is putting into HDFS.
9. Once the Flume agent has finished, enter Ctrl+C to terminate the process.

10 .Using the command line or Hue File Browser, list the files that were added by the Flume agent in the HDFS directory `/loudacre/weblogs_flume`.

Note that the files that were imported are tagged with a Unix timestamp corresponding to the time the file was imported, such as FlumeData.1504957031859.

```
[cloudera@quickstart flume]$ hdfs dfs -ls /loudacre/weblogs_flume | head -20
Found 622 items
-rw-r--r--   1 cloudera supergroup     527886 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031859
-rw-r--r--   1 cloudera supergroup     527781 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031860
-rw-r--r--   1 cloudera supergroup     527775 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031861
-rw-r--r--   1 cloudera supergroup     527861 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031862
-rw-r--r--   1 cloudera supergroup     527848 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031863
-rw-r--r--   1 cloudera supergroup     527890 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031864
-rw-r--r--   1 cloudera supergroup     527858 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031865
-rw-r--r--   1 cloudera supergroup     527834 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031866
-rw-r--r--   1 cloudera supergroup     527847 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031867
-rw-r--r--   1 cloudera supergroup     527807 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031868
-rw-r--r--   1 cloudera supergroup     527840 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031869
-rw-r--r--   1 cloudera supergroup     527789 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031870
-rw-r--r--   1 cloudera supergroup     527879 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031871
-rw-r--r--   1 cloudera supergroup     527840 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031872
-rw-r--r--   1 cloudera supergroup     527807 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031873
-rw-r--r--   1 cloudera supergroup     527877 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031874
-rw-r--r--   1 cloudera supergroup     527924 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031875
-rw-r--r--   1 cloudera supergroup     527875 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031876
-rw-r--r--   1 cloudera supergroup     527885 2017-09-09 04:37 /loudacre/weblogs_flume/FlumeData.1504957031877
[cloudera@quickstart flume]$
```

Exercise 11: Send Web Server Log Messages from Apache Flume to Apache Kafka

Files and Data Used in This Exercise

Exercise directory: `/home/cloudera/training/devsh/exercises/flafka`

Data files (local): `/home/cloudera/training/data/weblogs/*`

```
[cloudera@quickstart flafka]$ pwd
/home/cloudera/training/devsh/exercises/flafka
[cloudera@quickstart flafka]$ ls -ltr
total 8
-rw-r--r-- 1 cloudera cloudera 884 Nov 14  2016 spooldir_kafka.conf
-rwxr-xr-x 1 cloudera cloudera 832 Sep  9 04:52 copy-move-weblogs.sh
[cloudera@quickstart flafka]$
```

In this exercise, you will run a Flume agent that ingests web logs from a local spool directory and sends each line as a message to a Kafka topic.

The Flume agent is configured to send messages to the weblogs topic you created earlier.

Important: This exercise depends on two prior exercises: "Collect Web Server Logs with Flume" and "Produce and Consume Kafka Messages."

Configuring a Flume Agent with a Kafka Sink

A Flume agent configuration file has been provided for you:

`/home/cloudera/training/devsh/exercises/flafka/spooldir_kafka.conf`

1. Review the configuration file. You do not need to edit this file. Take note in particular of the following points:

The source and channel configurations are identical to the ones in the "Collect Web Server Logs with Flume" exercise: a spooling directory source that pulls from the local `/flume/weblogs_spooldir` directory, and a memory channel.

Instead of an HDFS sink, this configuration uses a Kafka sink that publishes messages to the weblogs topic.

Running the Flume Agent

2. Change to the exercise directory:

```
$ cd /home/cloudera/training/devsh/exercises/flafka
```

3. Start the Flume agent using the configuration you just reviewed:

```
$ flume-ng agent --conf /etc/flume-ng/conf \ --conf-file
spooldir_kafka.conf \  --name agent1 -Dflume.root.logger=INFO,console
```

```
[cloudera@quickstart flafka]$
[cloudera@quickstart flafka]$ flume-ng agent --conf /etc/flume-ng/conf \
> --conf-file spooldir_kafka.conf \
> --name agent1 -Dflume.root.logger=INFO,console
Info: Sourcing environment configuration script /etc/flume-ng/conf/flume-env.sh
Info: Including Hadoop libraries found via (/usr/bin/hadoop) for HDFS access
Info: Including HBASE libraries found via (/usr/bin/hbase) for HBASE access
Info: Including Hive libraries found via () for Hive access
```

4. Wait a few moments for the Flume agent to start up. You will see a message like:
   `Component type: SINK, name: kafka-sink started`

```
2017-09-09 04:59:26,376 (lifecycleSupervisor-1-1) [INFO - org.apache.flume.instrumentation.MonitoredCounterGroup.register
(MonitoredCounterGroup.java:119)] Monitored counter group for type: SINK, name: kafka-sink: Successfully registered new M
Bean.
2017-09-09 04:59:26,376 (lifecycleSupervisor-1-1) [INFO - org.apache.flume.instrumentation.MonitoredCounterGroup.start(Mo
nitoredCounterGroup.java:95)] Component type: SINK, name: kafka-sink started
```

Tip: This exercise involves using multiple terminal windows. To avoid confusion, set a different title for each window. Set the title of the current window to "Flume Agent."

Testing the Flume Agent Kafka Sink

5. In a new terminal window, start a Kafka consumer that will read from the weblogs topic:

Navigate to the path where Kafka is installed.

```
$ kafka-console-consumer \ --zookeeper localhost:2181 \ --
topic weblogs
```

```
[cloudera@quickstart ~]$ kafka-console-consumer \
> --zookeeper localhost:2181 \
> --topic weblogs
```

Tip: Set the title of this window to "Kafka Consumer."

6. In a separate new terminal window, change to the exercise directory. Run the script to place the web log files in the `/flume/weblogs_spooldir` directory:

```
$ cd /home/cloudera/training/devsh/exercises/flafka
$ ./copy-move-weblogs.sh /flume/weblogs_spooldir
```

```
[cloudera@quickstart flume]$
[cloudera@quickstart flume]$ pwd
/home/cloudera/training/devsh/exercises/flume
[cloudera@quickstart flume]$ ls -ltr
total 8
-rw-r--r-- 1 cloudera cloudera 943 Nov 14  2016 spooldir.conf
-rwxr-xr-x 1 cloudera cloudera 832 Sep  9 04:22 copy-move-weblogs.sh
[cloudera@quickstart flume]$ ./copy-move-weblogs.sh /flume/weblogs_spooldir/
```

Note that if you completed an earlier Flume exercise, the script will prompt you whether you want to clear out the spooldir directory. Be sure to enter y when prompted.

```
[cloudera@quickstart flume]$ ./copy-move-weblogs.sh /flume/weblogs_spooldir/
/flume/weblogs_spooldir/ exists and is not empty, delete contents? (y/n)
```

7. In the terminal that is running the Flume agent, watch the logging output. The output will give information about the files Flume is ingesting from the source directory.

```
2017-09-09 05:06:51,544 (pool-3-thread-1) [INFO - org.apache.flume.client.avro.ReliableSpoolingFileEventReader.rollCurren
tFile(ReliableSpoolingFileEventReader.java:433)] Preparing to move file /flume/weblogs_spooldir/2014-02-23.log to /flume/
weblogs_spooldir/2014-02-23.log.COMPLETED
2017-09-09 05:06:51,712 (pool-3-thread-1) [INFO - org.apache.flume.client.avro.ReliableSpoolingFileEventReader.readEvents
(ReliableSpoolingFileEventReader.java:324)] Last read took us just up to a file boundary. Rolling to the next file, if th
ere is one.
2017-09-09 05:06:51,712 (pool-3-thread-1) [INFO - org.apache.flume.client.avro.ReliableSpoolingFileEventReader.rollCurren
tFile(ReliableSpoolingFileEventReader.java:433)] Preparing to move file /flume/weblogs_spooldir/2014-03-09.log to /flume/
weblogs_spooldir/2014-03-09.log.COMPLETED
2017-09-09 05:06:51,922 (pool-3-thread-1) [INFO - org.apache.flume.client.avro.ReliableSpoolingFileEventReader.readEvents
(ReliableSpoolingFileEventReader.java:324)] Last read took us just up to a file boundary. Rolling to the next file, if th
ere is one.
2017-09-09 05:06:51,923 (pool-3-thread-1) [INFO - org.apache.flume.client.avro.ReliableSpoolingFileEventReader.rollCurren
tFile(ReliableSpoolingFileEventReader.java:433)] Preparing to move file /flume/weblogs_spooldir/2014-03-12.log to /flume/
weblogs_spooldir/2014-03-12.log.COMPLETED
```

8. In the terminal that is running the Kafka consumer, confirm that the consumer tool is displaying each message (that is, each line of the web log file Flume is ingesting).

```
80.53.185.159 - 25704 [16/Dec/2013:20:49:44 +0100] "GET /discounts.html HTTP/1.0" 200 15302 "http://www.loudacre.com"  "Lo
udacre Mobile Browser Ronin Novelty Note 1"
80.53.185.159 - 25704 [16/Dec/2013:20:49:44 +0100] "GET /theme.css HTTP/1.0" 200 8182 "http://www.loudacre.com"  "Loudacre
 Mobile Browser Ronin Novelty Note 1"
80.53.185.159 - 25704 [16/Dec/2013:20:49:44 +0100] "GET /code.js HTTP/1.0" 200 18242 "http://www.loudacre.com"  "Loudacre
Mobile Browser Ronin Novelty Note 1"
80.53.185.159 - 25704 [16/Dec/2013:20:49:44 +0100] "GET /promo.jpg HTTP/1.0" 200 2785 "http://www.loudacre.com"  "Loudacre
 Mobile Browser Ronin Novelty Note 1"
150.176.131.129 - 74 [16/Dec/2013:20:49:33 +0100] "GET /KBDOC-00286.html HTTP/1.0" 200 9672 "http://www.loudacre.com"  "Lo
udacre CSR Browser"
150.176.131.129 - 74 [16/Dec/2013:20:49:33 +0100] "GET /theme.css HTTP/1.0" 200 16730 "http://www.loudacre.com"  "Loudacre
 CSR Browser"
165.80.36.248 - 44110 [16/Dec/2013:20:49:31 +0100] "GET /KBDOC-00042.html HTTP/1.0" 200 13173 "http://www.loudacre.com"  "
Loudacre Mobile Browser iFruit 2"
```

9. Once the Flume agent has finished, enter Ctrl+C in both the Flume agent terminal and the Kafka consumer terminal to end their respective processes.

Exercise12: Write an Apache Spark Streaming Application

In this exercise, you will write a Spark Streaming application to count Knowledge Base article requests.

This exercise has two parts. First, you will review the Spark Streaming documentation. Then you will write and test a Spark Streaming application to read streaming web server log data and count the number of requests for Knowledge Base articles.

Reviewing the Spark Streaming Documentation

1. View the Spark Streaming API by opening the Spark API documentation for either Scala or Python and then:

For Scala:

Scroll down and select the org.apache.spark.streaming package in the package pane on the left.

Follow the links at the top of the package page to view the DStream and PairDStreamFunctions classes— these will show you the methods available on a DStream of regular RDDs and pair RDDs respectively.

For Python:

Go to the `pyspark.streaming` module.

Scroll down to the `pyspark.streaming.DStream` class and review the available methods.

2. You may also wish to view the Spark Streaming Programming Guide (select Programming Guides > Spark Streaming on the Spark documentation main page).

Simulating Streaming Web Logs

To simulate a streaming data source, you will use the provided streamtest.py Python script, which waits for a connection on the host and port specified and, once it receives a connection, sends the contents of the file(s) specified to the client (which will be your Spark Streaming application). You can specify the speed (in lines per second) at which the data should be sent.

3. Change to the exercise directory.

```
$ cd /home/cloudera/training/devsh/exercises/spark-streaming
```

```
[cloudera@quickstart spark-streaming]$ pwd
/home/cloudera/training/devsh/exercises/spark-streaming
[cloudera@quickstart spark-streaming]$ ls -ltr
total 16
drwxr-xr-x 2 cloudera cloudera 4096 Nov 14  2016 stubs-python
-rwxrwxr-x 1 cloudera cloudera 1383 Nov 14  2016 streamtest.py
drwxr-xr-x 3 cloudera cloudera 4096 Nov 14  2016 streaminglogs_project
drwxr-xr-x 2 cloudera cloudera 4096 Nov 14  2016 solution-python
[cloudera@quickstart spark-streaming]$
```

4. Stream the Loudacre web log files at a rate of 20 lines per second using the provided test script.

```
$ python streamtest.py localhost 1234 20
\ /home/cloudera/training/data/weblogs/*
```

```
[cloudera@quickstart spark-streaming]$ python streamtest.py localhost 1234 20 \
> /home/cloudera/training/data/weblogs/*
Waiting for connection on localhost : 1234
```

This script will exit after the client disconnects, so you will need to restart the script whenever you restart your Spark application.

Writing a Spark Streaming Application

5. To help you get started writing a Spark Streaming application, stub files have been provided for you.

   For Python, start with the stub file `StreamingLogs.py` in the `/home/cloudera/training/devsh/exercises/spark-streaming/stubs-python` directory, which imports the necessary classes for the application.

   For Scala, a Maven project directory called `streaminglogs_project` has been provided in the exercise directory `/home/cloudera/training/devsh/exercises/spark-streaming`).

   To complete the exercise, start with the stub code in `src/main/scala/stubs/StreamingLogs.scala`, which imports the necessary classes for the application.

6. Define a Streaming context with a one-second batch duration.

7. Create a DStream by reading the data from the host and port provided as input parameters.

8. Filter the DStream to only include lines containing the string KBDOC.

9. To confirm that your application is correctly receiving the streaming web log data, display the first five records in the filtered DStream for each one-second batch. (In Scala, use the DStream print function; in Python, use pprint.)

10. For each RDD in the filtered DStream, display the number of items—that is, the

number of requests for KB articles.

Tip: Python does not allow calling print within a lambda function, so create a named defined function to print.

11. Save the filtered logs to text files in HDFS. Use the base directory name `/loudacre/streamlog/kblogs`.

12. Finally, start the Streaming context, and then call `awaitTermination()`.

Testing the Application

13. In a new terminal window, change to the correct directory for the language you are using for your application.

    For Python, change to the exercise directory:

```
$ cd /home/cloudera/training/devsh/exercises/spark-streaming
```

    For Scala, change to the project directory for the exercise:

```
$ cd \
$/home/cloudera/training/devsh/exercises/spark-
streaming/streaminglogs_project
```

14. If you are using Scala, build your application JAR file using the mvn package command.
15. Use spark-submit to run your application locally and be sure to specify two threads; at least two threads or nodes are required to run a streaming application, while the VM cluster has only one.

    The StreamingLogs application takes two parameters: the host name and the port number to connect the DStream to. Specify the same host and port at which the test script you started earlier is listening.

```
$ spark-submit --master 'local[2]' \ solution-
python/StreamingLogs.py localhost 1234
```

```
[cloudera@quickstart solution-python]$ python streamtest.py localhost 1234 20 /home/cloudera/training/data/weblogs/*
python: can't open file 'streamtest.py': [Errno 2] No such file or directory
[cloudera@quickstart solution-python]$ spark-submit --master 'local[2]' StreamingLogs.py localhost 1234
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/parquet/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/avro/avro-tools-1.7.6-cdh5.12.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
17/09/09 20:01:15 INFO spark.SparkContext: Running Spark version 1.6.0
```

```
$ spark-submit --master 'local[2]' \ --class
solution.StreamingLogs \ target/streamlog-1.0.jar localhost 1234
```

16. After a few moments, the application should connect to the test script's simulated stream of web server log output. Confirm that for every batch of data received (every second), the application displays the first few Knowledge Base requests and the count of requests in the batch. Review the HDFS files the application saved in `/loudacre/streamlog`.

17. Return to the terminal window in which you started the streamtest.py test script earlier. Stop the test script by typing Ctrl+C. You do not need to wait until all the web log data has been sent

```
[cloudera@quickstart spark-streaming]$ python streamtest.py localhost 1234 20 /home/cloudera/training/data/weblogs/*
Waiting for connection on localhost : 1234
Connection from ('127.0.0.1', 36149)
Sending /home/cloudera/training/data/weblogs/2013-09-15.log
3.94.78.5 - 69827 [15/Sep/2013:23:58:36 +0100] "GET /KBDOC-00033.html HTTP/1.0" 200 14417 "http://www.loudacre.com"  "Loudacre Mobile Browser iFruit 1"

3.94.78.5 - 69827 [15/Sep/2013:23:58:36 +0100] "GET /theme.css HTTP/1.0" 200 3576 "http://www.loudacre.com"  "Loudacre Mobile Browser iFruit 1"

19.38.140.62 - 21475 [15/Sep/2013:23:58:34 +0100] "GET /KBDOC-00277.html HTTP/1.0" 200 15517 "http://www.loudacre.com"  "Loudacre Mobile Browser Ronin S1"

19.38.140.62 - 21475 [15/Sep/2013:23:58:34 +0100] "GET /theme.css HTTP/1.0" 200 13353 "http://www.loudacre.com"  "Loudacre Mobile Browser Ronin S1"

129.133.56.105 - 2489 [15/Sep/2013:23:58:34 +0100] "GET /KBDOC-00033.html HTTP/1.0" 200 10590 "http://www.loudacre.com"  "Loudacre Mobile Browser Sorrento F0
0L"

129.133.56.105 - 2489 [15/Sep/2013:23:58:34 +0100] "GET /theme.css HTTP/1.0" 200 12295 "http://www.loudacre.com"  "Loudacre Mobile Browser Sorrento F00L"

217.150.149.167 - 4712 [15/Sep/2013:23:56:06 +0100] "GET /ronin_s4_sales.html HTTP/1.0" 200 845 "http://www.loudacre.com"  "Loudacre Mobile Browser MeeToo 1.
0"

217.150.149.167 - 4712 [15/Sep/2013:23:56:06 +0100] "GET /theme.css HTTP/1.0" 200 738 "http://www.loudacre.com"  "Loudacre Mobile Browser MeeToo 1.0"
```

18. After the test script has stopped, stop your application by typing Ctrl+C in the terminal window the application is running in.

Exercise13: Process Multiple Batches with Apache Spark Streaming

In this exercise, you will write a Spark Streaming application to count web page requests over time.

Simulating Streaming Web Logs

To simulate a streaming data source, you will use the provided streamtest.py Python script, which waits for a connection on the host and port specified and, once it receives a connection, sends the contents of the file(s) specified to the client (which will be your Spark Streaming application). You can specify the speed (in lines per second) at which the data should be sent.

12. Change to the exercise directory.

```
$ cd /home/cloudera/training/devsh/exercises/spark-streaming-multi
```

2. Stream the Loudacre Web log files at a rate of 20 lines per second using the provided test script.

```
$ python streamtest.py localhost 1234 20 \

$ /home/cloudera/training/data/weblogs/*
```

This script exits after the client disconnects, so you will need to restart the script when you restart your Spark application.

Displaying the Total Request Count

3. A stub file for this exercise has been provided for you in the exercise directory.

The stub code creates a Streaming context for you, and creates a DStream called logs based on web log request messages received on a network socket.

For Python, start with the stub file `StreamingLogsMB.py` in the `stubs- python` directory.

For Scala, a Maven project directory called `streaminglogsMB_project` has been provided in the exercise directory. To complete the exercise, start with the stub code in `src/main/scala/stubs/StreamingLogsMB.scala`.

4. Enable checkpointing to a directory called logcheckpt.
5. Count the number of page requests over a window of five seconds. Print out the updated five-second total every two seconds.

Hint: Use the countByWindow function.

Building and Running Your Application

6. In a different terminal window than the one in which you started the streamtest.py script, change to the correct directory for the language you are using for your application.

For Python, change to the exercise directory:

```
$ cd /home/cloudera/training/devsh/exercises/spark-streaming-multi
```

For Scala, change to the project directory for the exercise:

```
$ cd \
$ /home/cloudera/training/devsh/exercises/spark-streaming-
multi/streaminglogsMB_project
```

If you are using Scala, build your application JAR file using the mvn package command.

Use spark-submit to run your application locally and be sure to specify two threads; at least two threads or nodes are required to running a streaming application, while the VM cluster has only one. Your application takes two parameters: the host name and the port number to connect the DStream to. Specify the same host and port at which the test script you started earlier is listening.

Exercise 14: Integration with Spark Streaming

Users leave thousands of traces per second on a successful ecommerce site. It's very pragmatic to analyse and react on this trace event stream in realtime. This is called clickstream analysis.

We are going to analyse the clickstream generated continually from an ecommerce site. The building blocks of the architecture beside Spark are Kafka to handle the inbound event stream.

Let us create a SparkStreaming application which is analysing the data collected from a clickstream. The data will be extracted for the various URLs and for every five minutes, the data is aggregated for the URL and is sorted and printed out in the console.

1. Create a maven project with the groupId and artifactId.
2. Update the maven with the dependencies related to Spark. The project is created for the lab set and can be located under "/home/cloudera/workspace/KafkaStreamSpark".
3. The Streaming app "KafkaExample" is added as Scala object which is windowed for 5 minutes.
4. Export the project as Jar file in the location "/home/cloudera/workspace/KafkaStream"
5. Submit the jar file using spark-submit

```
spark-submit --class "KafkaExample"
/home/cloudera/workspace/KafkaStream/streamapp.jar
```

6. To Test the application,
7. Create the topic in Kafka where we will push the clickstream generated as log files as testlogs.

```
[cloudera@quickstart kafka]$ kafka-topics --create --zookeeper localhost:2181 --
replication-factor 1 --partitions 1 --topic testlogs
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Created topic "testlogs".
```

8. Push the messages in the Kafka broker from kafka producer with the script

```
kafka-console-producer --broker-list localhost:9092 --topic testLogs
< /home/cloudera/workspace/KafkaStreamSpark/access_log.txt
```

```
[cloudera@quickstart kafka]$ kafka-console-producer --broker-list localhost:9092 --topic testLogs < /home/cloudera/workspace/KafkaStreamSpark/access_log.txt
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
```

9. The logs file is being read from the Kafka topic, will start streaming in the console and is analysed for the webURL specified, as below

```
-------------------------------------------
Time: 1520622374000 ms
-------------------------------------------
(/xmlrpc.php,11820)
(/wp-login.php,1799)
(/,130)
(/robots.txt,48)
(/blog/,38)
(/sitemap_index.xml,32)
(/category-sitemap.xml,31)
(/post-sitemap.xml,31)
(/page-sitemap.xml,31)
(http://51.254.206.142/httptest.php,29)
...

-------------------------------------------
Time: 1520622375000 ms
-------------------------------------------
(/xmlrpc.php,11820)
(/wp-login.php,1799)
(/,130)
(/robots.txt,48)
(/blog/,38)
(/sitemap_index.xml,32)
(/category-sitemap.xml,31)
(/post-sitemap.xml,31)
(/page-sitemap.xml,31)
(http://51.254.206.142/httptest.php,29)
...
```