Hands on Exercises for Cloudera Hadoop-Developer Track

Table of Contents

Exercise1: Setting up Environment 3

Exercise 2: Practicing HDFS Commands 7

Exercise 3: Spark 14

Exercise 4: Operations On Multiple RDDs 32

Exercise5: Building Spark Application using IntelliJ IDE 39

Exercise 6: Accumulators 42

.... 43

Exercise 7: Broadcast variables 44

Exercise 8: Dataframe, Datasets and Spark SQL 46

Exercise 9: Dataframe, Datasets and Spark SQL Exploring Grouping 69

Exercise 10: Seeing Catalyst at Work 71

.... 74

Exercise 11: Joins and Broadcast 74

.... 78

Exercise 12: Hive with Spark integration 79

Exercise 13: SQL Tables and Views 81

.... 83

Exercise 14: Hive with Spark integration using IntelliJ 83

Exercise 15: Produce and Consume Apache Kafka Messages 88

Exercise 16: Alter Apache Kafka Topics 95

Exercise 17: Spark Streaming 116

Exercise 18: Apache Kafka Connector Example – Import Data from MySQL into Kafka 118

Exercise 19: Apache Kafka Connector Example – Kafka Connect JDBC Sink 124

.... 126

Exercise 20: Add Scala Kernal in Jupyter 126

Exercise1: Setting up Environment

Step1: Installing VMware Player from Training Bundle.

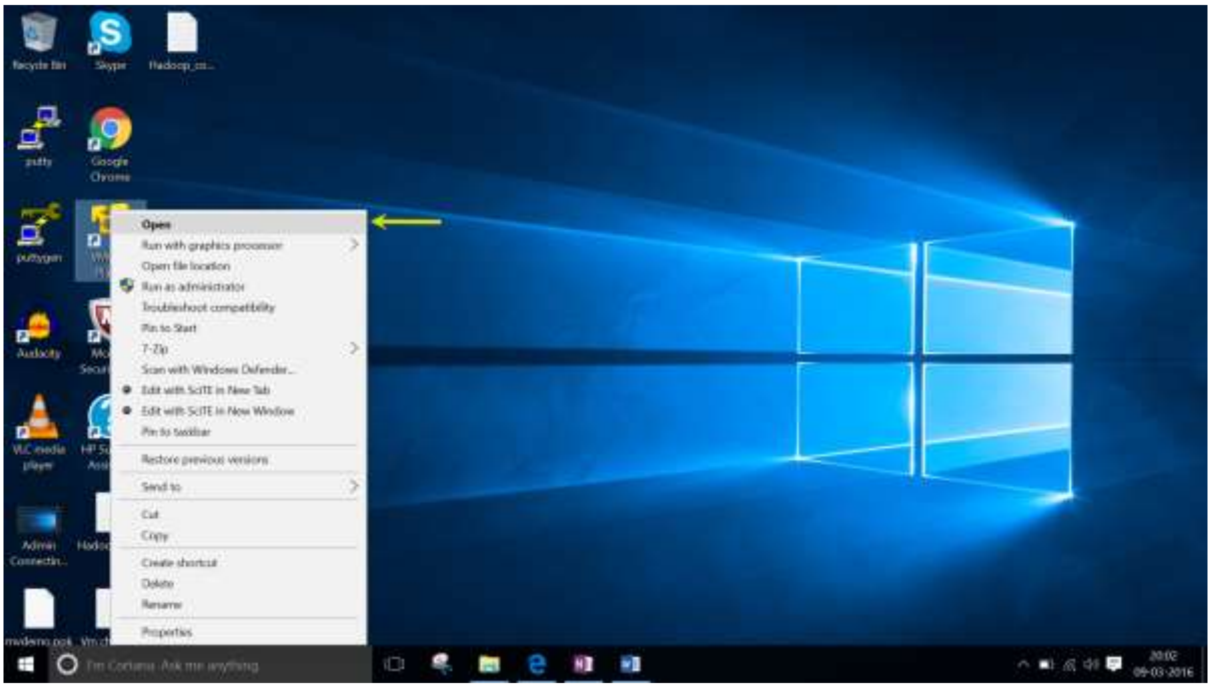
- Under 'Training_Bundle_For_Hortonworks_Developer_Track' folder, navigate to "Additional Software" folder.
- Find the executable file named 'VmWarePlayer.exe' and complete the installation.

Step2: Extracting the VMware Image

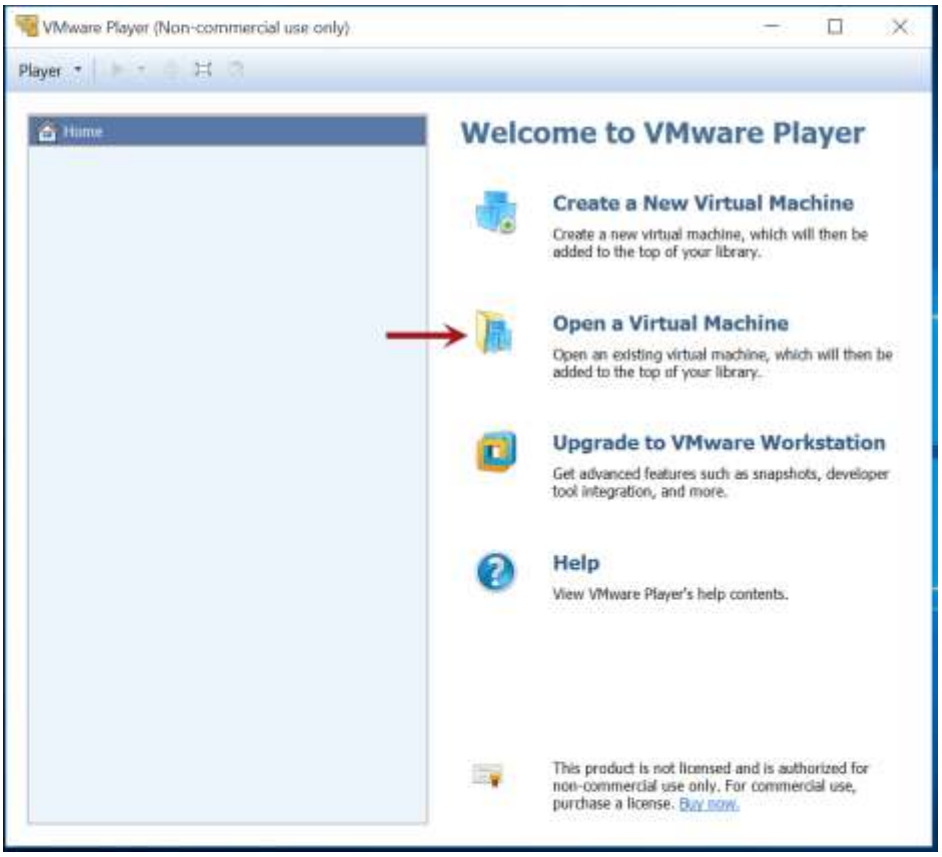
- From the same training bundle, locate the folder named VM image> 'technocrafty-quickstart-vm-5.5.0-0-vmware'
- Unzip/Extract the files and using the VMware player browse to this location to get start with virtual machine.

Step3: Loading the VMware Image

- Right click and select Open or Double click on 'VMware Player' and launch the same.


Open
Run with graphics processor
Open file location
Run as administrator
Troubleshoot compatibility
Pin to Start
7-Zip
Scan with Windows Defender...
Edit with SciTE in New Tab
Edit with SciTE in New Window
Pin to taskbar
Restore previous versions
Send to
Cut
Copy
Create shortcut
Delete
Rename
Properties

- Select the option as "Open a Virtual Machine"



Welcome to VMware Player

Create a New Virtual Machine
Create a new virtual machine, which will then be added to the top of your library.

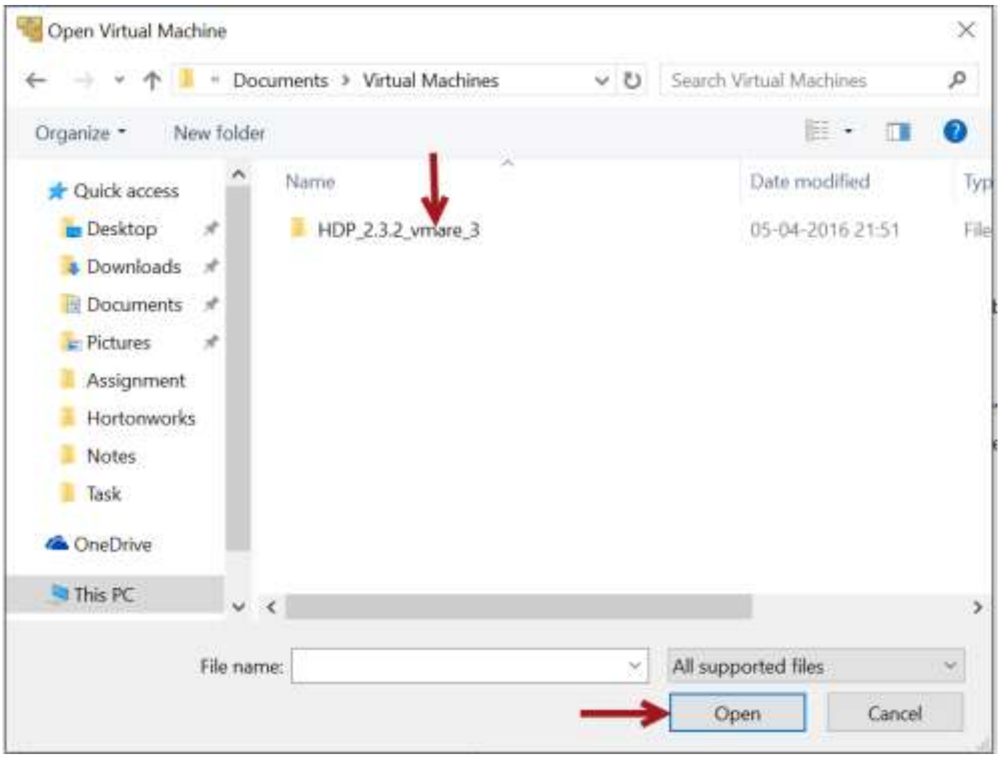
Open a Virtual Machine
Open an existing virtual machine, which will then be added to the top of your library.

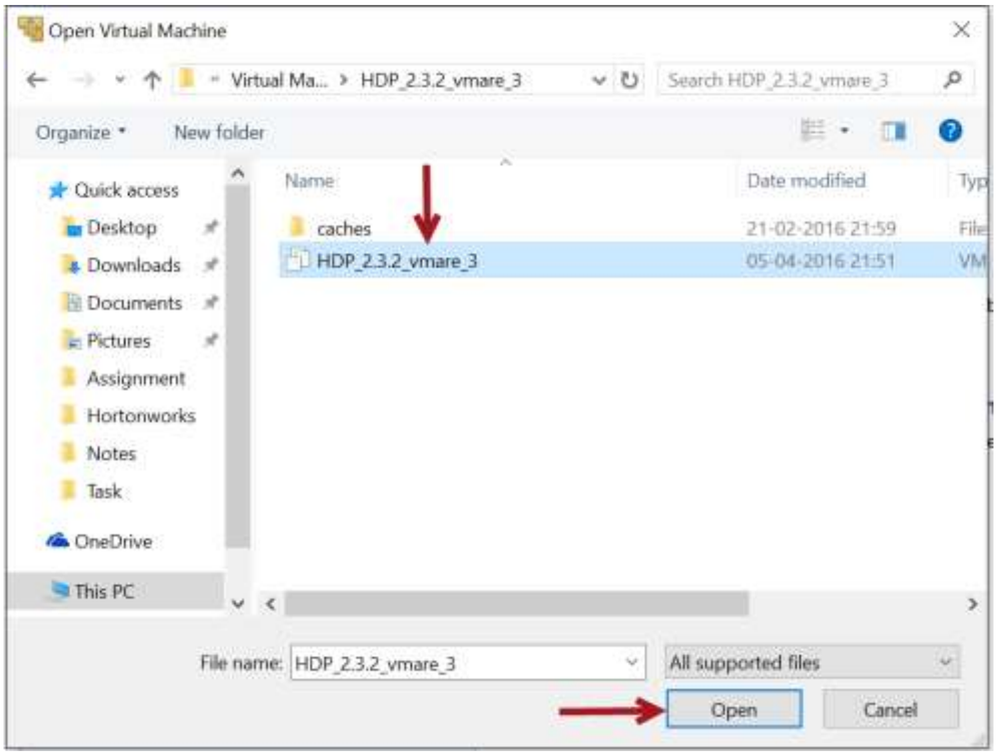
Upgrade to VMware Workstation
Get advanced features such as snapshots, developer tool integration, and more.

Help
View VMware Player's help contents.

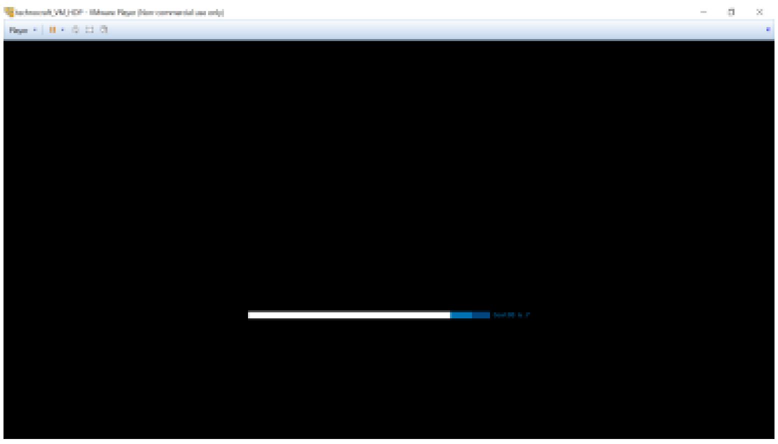
This product is not licensed and is authorized for non-commercial use only. For commercial use, purchase a license. Buy now.

- Navigate to the location where you have extracted the VMware Image in earlier step.


Open Virtual Machine
Documents > Virtual Machines
Search Virtual Machines
Organize
New folder
Quick access
Desktop
Downloads
Documents
Pictures
Assignment
Hortonworks
Notes
Task
OneDrive
This PC
Name
Date modified
Typ
HDP_2.3.2_vmare_3
05-04-2016 21:51
File
File name:
All supported files
Open
Cancel


Open Virtual Machine
Virtual Ma... > HDP_2.3.2_vmare_3
Search HDP_2.3.2_vmare_3
Organize
New folder
Quick access
Desktop
Downloads
Documents
Pictures
Assignment
Hortonworks
Notes
Task
OneDrive
This PC
Name
Date modified
Typ
caches
21-02-2016 21:59
File
HDP_2.3.2_vmare_3
05-04-2016 21:51
VM
File name: HDP_2.3.2_vmare_3
All supported files
Open
Cancel

- This will start the Virtual Machine



Login with user ‘cloudera’ with password ‘cloudera’

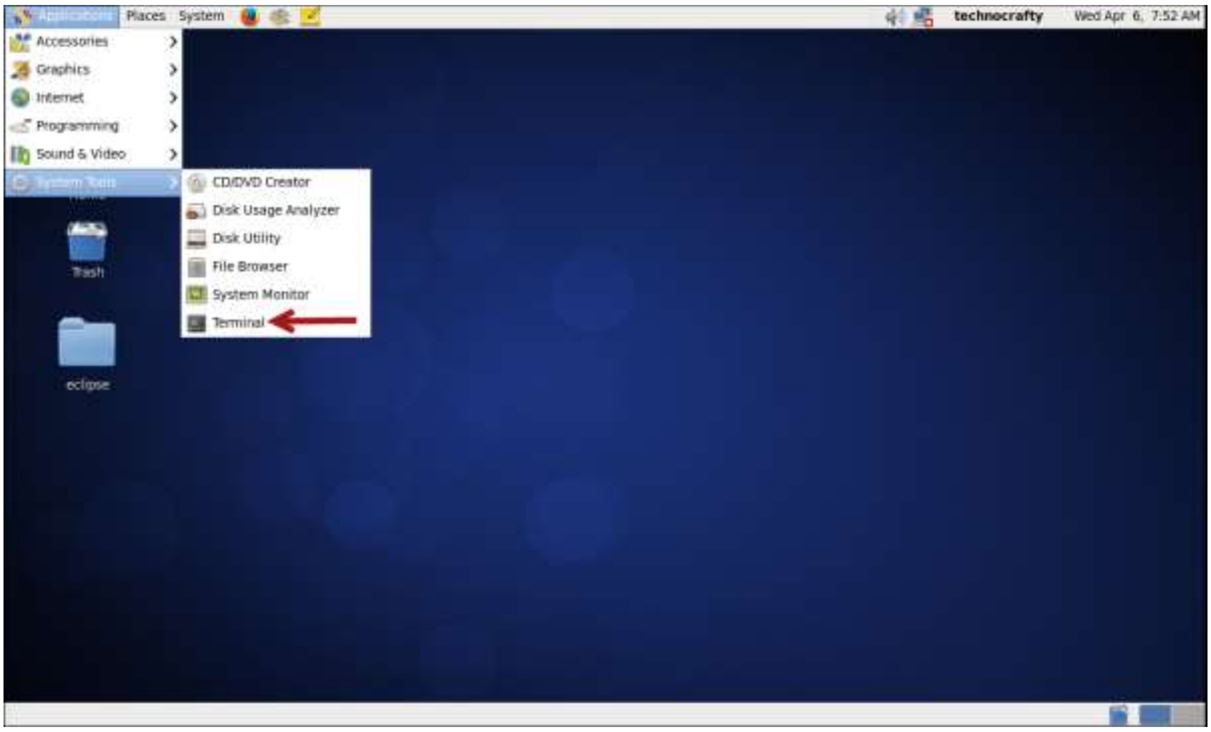
Exercise 2: Practicing HDFS Commands

We will practise HDFS command line interface and web-based Hue File Browser to perform operations on files.

HDFS Command line Interface

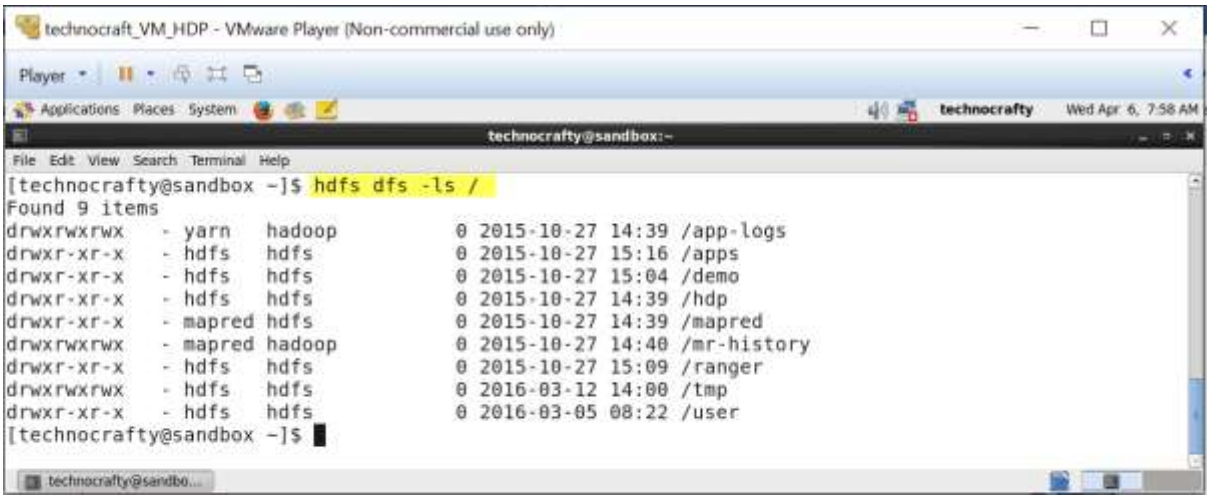
- Open terminal

Navigate to Application > System Tools > Terminal



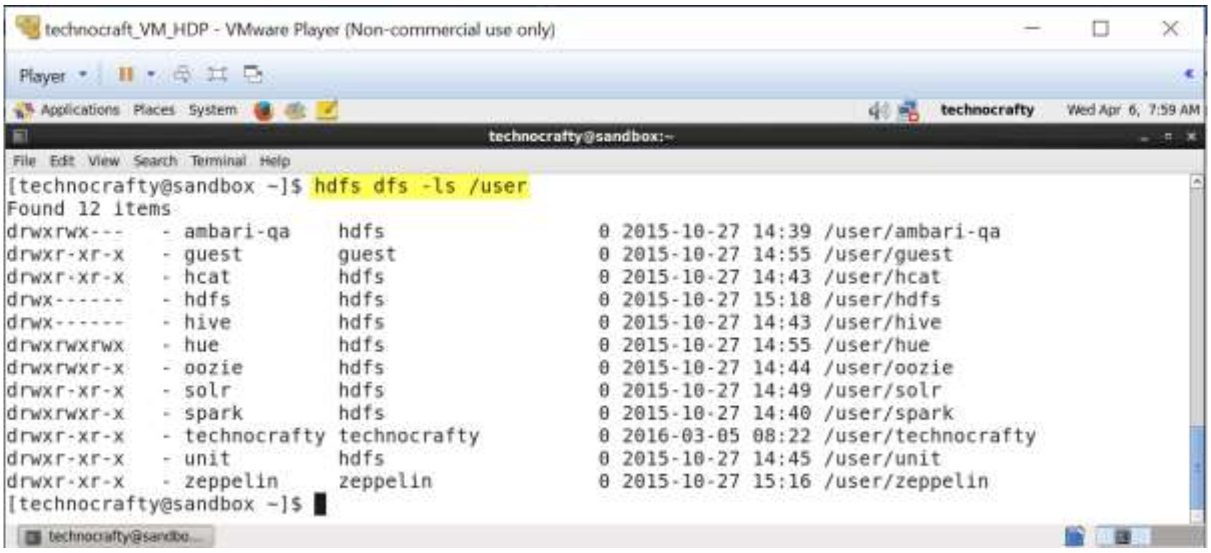
Operations on Files and Directories:

To view the content of root directory in HDFS


technocraft_VM_HDP - VMware Player (Non-commercial use only)
Player
Applications Places System
technocrafty Wed Apr 6, 7:58 AM
technocrafty@sandbox:~
File Edit View Search Terminal Help

```
[technocrafty@sandbox ~]$ hdfs dfs -ls /
Found 9 items
drwxrwxrwx   - yarn   hadoop          0 2015-10-27 14:39 /app-logs
drwxr-xr-x   - hdfs   hdfs            0 2015-10-27 15:16 /apps
drwxr-xr-x   - hdfs   hdfs            0 2015-10-27 15:04 /demo
drwxr-xr-x   - hdfs   hdfs            0 2015-10-27 14:39 /hdp
drwxr-xr-x   - mapred hdfs            0 2015-10-27 14:39 /mapred
drwxrwxrwx   - mapred hadoop          0 2015-10-27 14:40 /mr-history
drwxr-xr-x   - hdfs   hdfs            0 2015-10-27 15:09 /ranger
drwxrwxrwx   - hdfs   hdfs            0 2016-03-12 14:00 /tmp
drwxr-xr-x   - hdfs   hdfs            0 2016-03-05 08:22 /user
[technocrafty@sandbox ~]$ 
```

To view the content of /user directory


technocraft_VM_HDP - VMware Player (Non-commercial use only)
Player
Applications Places System
technocrafty Wed Apr 6, 7:59 AM
technocrafty@sandbox:~
File Edit View Search Terminal Help

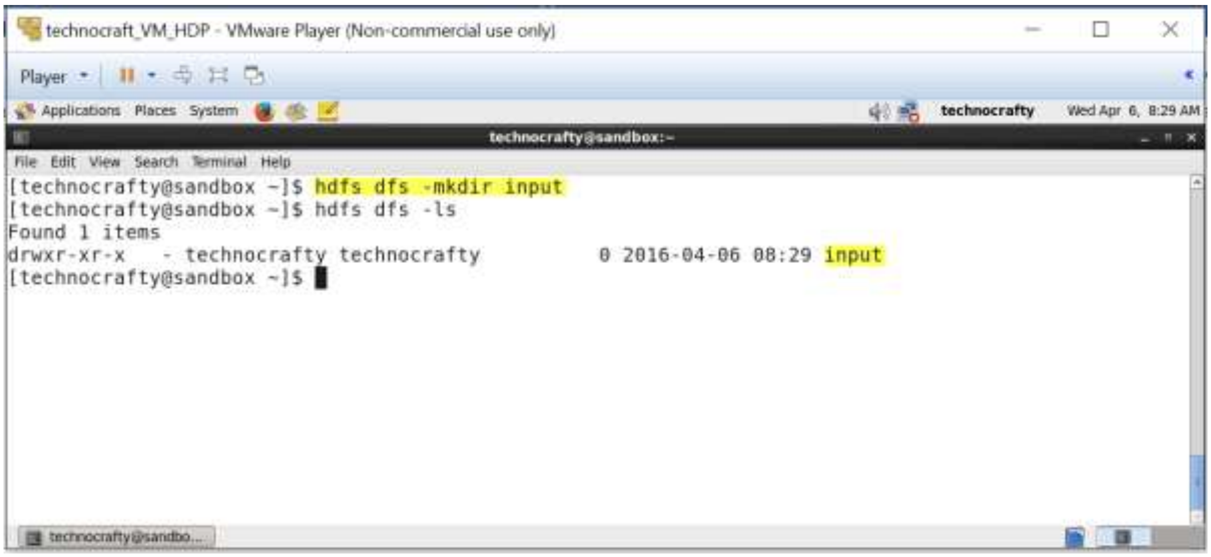
```
[technocrafty@sandbox ~]$ hdfs dfs -ls /user
Found 12 items
drwxrwx---   - ambari-qa    hdfs                0 2015-10-27 14:39 /user/ambari-qa
drwxr-xr-x   - guest        guest               0 2015-10-27 14:55 /user/guest
drwxr-xr-x   - hcat         hdfs                0 2015-10-27 14:43 /user/hcat
drwx------   - hdfs         hdfs                0 2015-10-27 15:18 /user/hdfs
drwx------   - hive         hdfs                0 2015-10-27 14:43 /user/hive
drwxrwxrwx   - hue          hdfs                0 2015-10-27 14:55 /user/hue
drwxrwxr-x   - oozie        hdfs                0 2015-10-27 14:44 /user/oozie
drwxr-xr-x   - solr         hdfs                0 2015-10-27 14:49 /user/solr
drwxrwxr-x   - spark        hdfs                0 2015-10-27 14:40 /user/spark
drwxr-xr-x   - technocrafty technocrafty        0 2016-03-05 08:22 /user/technocrafty
drwxr-xr-x   - unit         hdfs                0 2015-10-27 14:45 /user/unit
drwxr-xr-x   - zeppelin     zeppelin            0 2015-10-27 15:16 /user/zeppelin
[technocrafty@sandbox ~]$ 
```

Note

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.

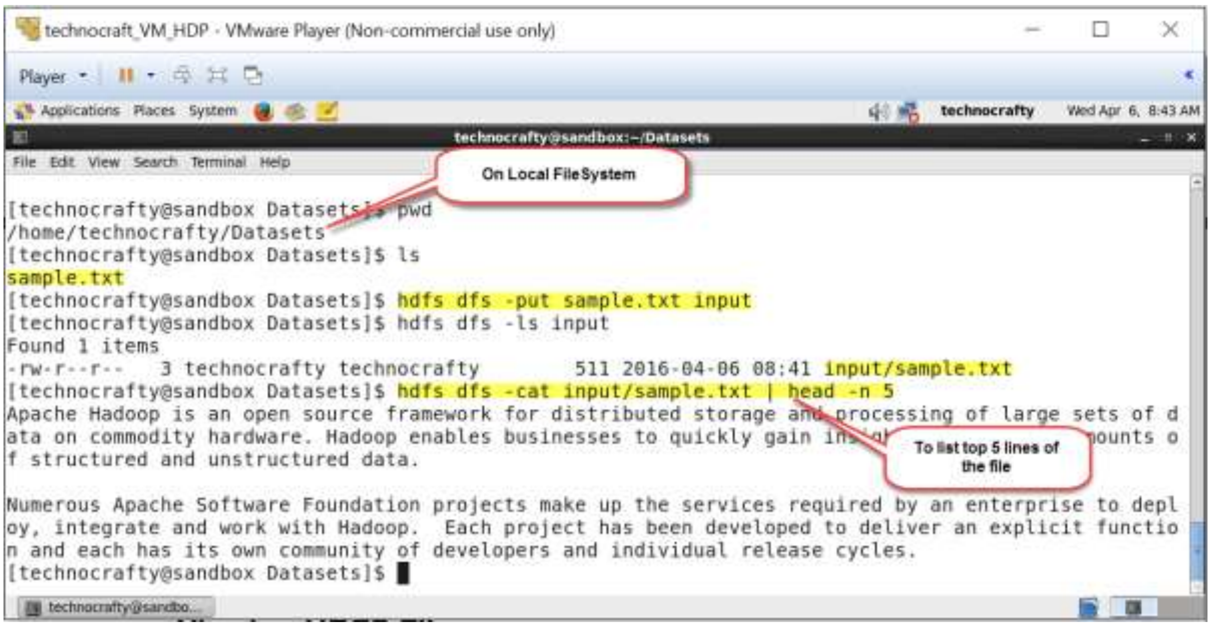
Uploading Files to HDFS

- Create a directory named 'input' in HDFS, we will use this directory throughout the exercises.
- Ensure that directory is created by running ls command


technocraft_VM_HDP - VMware Player (Non-commercial use only)
Applications Places System
technocrafty Wed Apr 6, 8:29 AM
technocrafty@sandbox:~
File Edit View Search Terminal Help

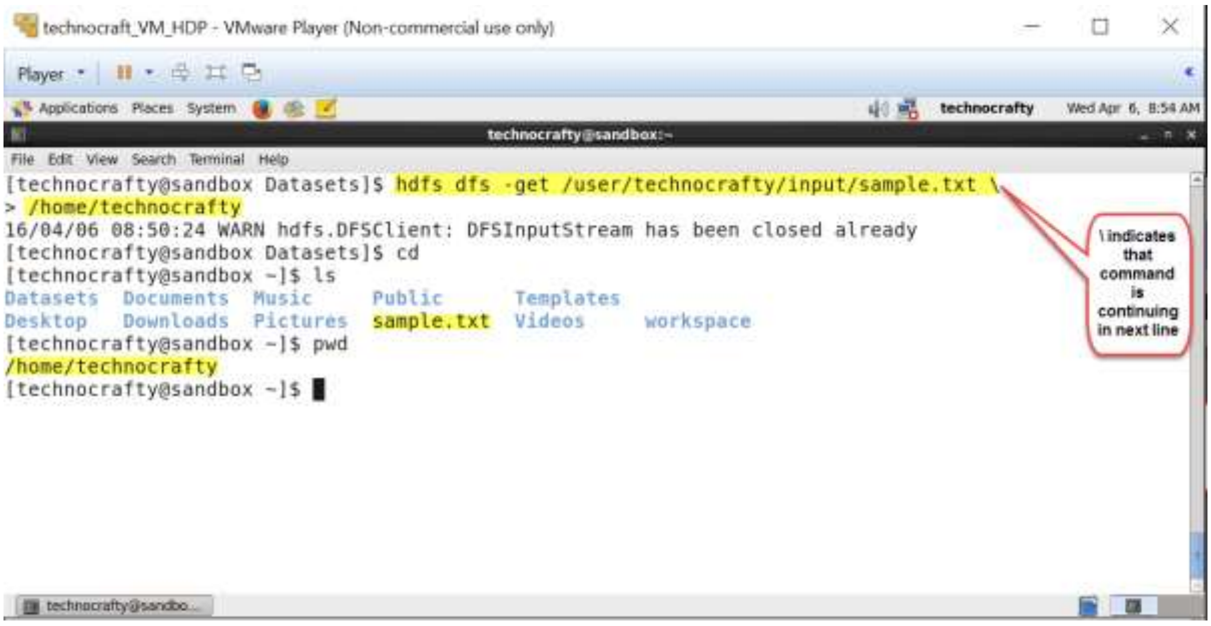
```
[technocrafty@sandbox ~]$ hdfs dfs -mkdir input
[technocrafty@sandbox ~]$ hdfs dfs -ls
Found 1 items
drwxr-xr-x   - technocrafty technocrafty          0 2016-04-06 08:29 input
[technocrafty@sandbox ~]$
```

➢ Locate test file named 'sample.txt' under Datasets directory on local filesystem and upload into HDFS


technocraft_VM_HDP - VMware Player (Non-commercial use only)
Applications Places System
technocrafty Wed Apr 6, 8:43 AM
technocrafty@sandbox:~/Datasets
File Edit View Search Terminal Help
On Local FileSystem
To list top 5 lines of the file

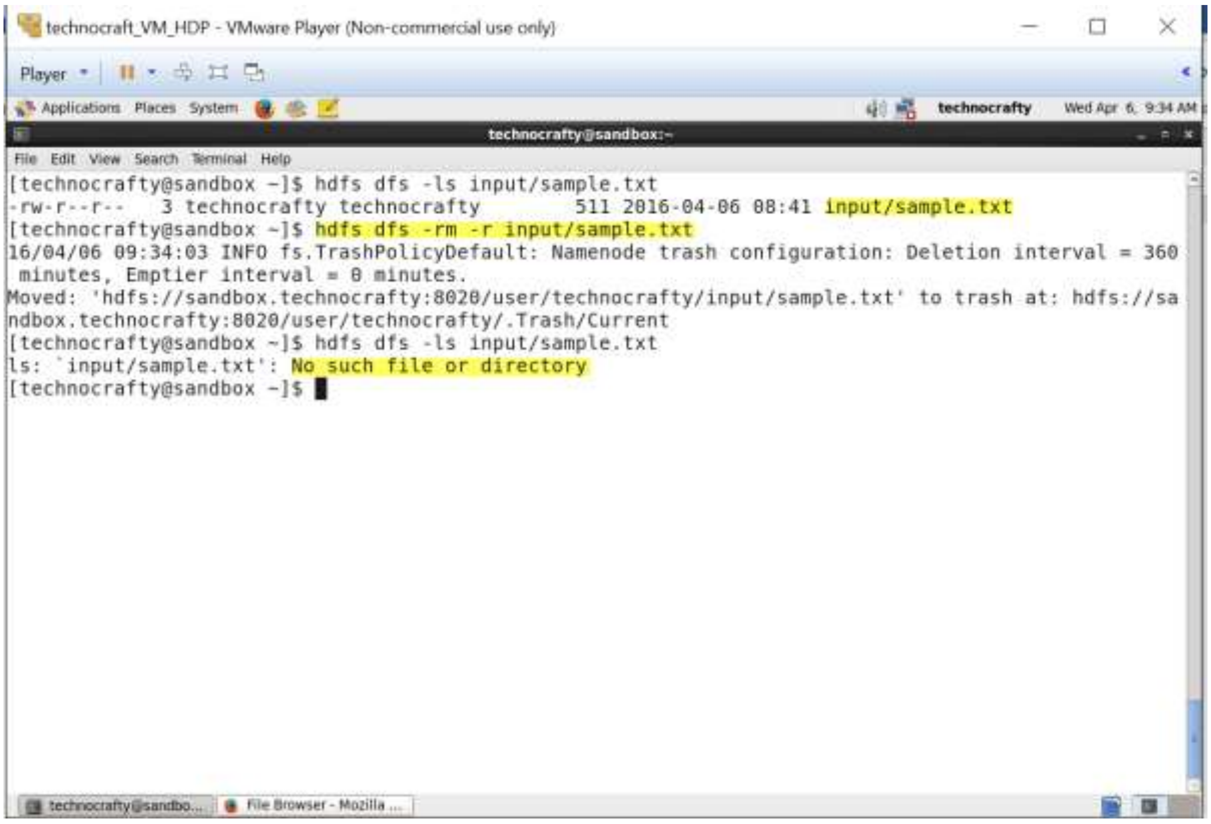
```
[technocrafty@sandbox Datasets]$ pwd
/home/technocrafty/Datasets
[technocrafty@sandbox Datasets]$ ls
sample.txt
[technocrafty@sandbox Datasets]$ hdfs dfs -put sample.txt input
[technocrafty@sandbox Datasets]$ hdfs dfs -ls input
Found 1 items
-rw-r--r--   3 technocrafty technocrafty        511 2016-04-06 08:41 input/sample.txt
[technocrafty@sandbox Datasets]$ hdfs dfs -cat input/sample.txt | head -n 5
Apache Hadoop is an open source framework for distributed storage and processing of large sets of d
ata on commodity hardware. Hadoop enables businesses to quickly gain ins[illegible] amounts o
f structured and unstructured data.

Numerous Apache Software Foundation projects make up the services required by an enterprise to depl
oy, integrate and work with Hadoop.  Each project has been developed to deliver an explicit functio
n and each has its own community of developers and individual release cycles.
[technocrafty@sandbox Datasets]$
```

➢ To download the file from HDFS to local filesystem


technocraft_VM_HDP - VMware Player (Non-commercial use only)
Player
Applications Places System
technocrafty
Wed Apr 6, 8:54 AM
technocrafty@sandbox:~
File Edit View Search Terminal Help
\ indicates that command is continuing in next line
technocrafty@sandbo...

```
[technocrafty@sandbox Datasets]$ hdfs dfs -get /user/technocrafty/input/sample.txt \
> /home/technocrafty
16/04/06 08:50:24 WARN hdfs.DFSClient: DFSInputStream has been closed already
[technocrafty@sandbox Datasets]$ cd
[technocrafty@sandbox ~]$ ls
Datasets  Documents  Music     Public      Templates
Desktop   Downloads  Pictures  sample.txt  Videos     workspace
[technocrafty@sandbox ~]$ pwd
/home/technocrafty
[technocrafty@sandbox ~]$ 
```

- To remove the file from HDFS


technocraft_VM_HDP - VMware Player (Non-commercial use only)
Player
Applications Places System
technocrafty
Wed Apr 6, 9:34 AM
technocrafty@sandbox:~
File Edit View Search Terminal Help
technocrafty@sandbo...
File Browser - Mozilla ...

```
[technocrafty@sandbox ~]$ hdfs dfs -ls input/sample.txt
-rw-r--r--   3 technocrafty technocrafty        511 2016-04-06 08:41 input/sample.txt
[technocrafty@sandbox ~]$ hdfs dfs -rm -r input/sample.txt
16/04/06 09:34:03 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 360
 minutes, Emptier interval = 0 minutes.
Moved: 'hdfs://sandbox.technocrafty:8020/user/technocrafty/input/sample.txt' to trash at: hdfs://sa
ndbox.technocrafty:8020/user/technocrafty/.Trash/Current
[technocrafty@sandbox ~]$ hdfs dfs -ls input/sample.txt
ls: `input/sample.txt': No such file or directory
[technocrafty@sandbox ~]$ 
```

Web-Based Interface (Hue File Browser)

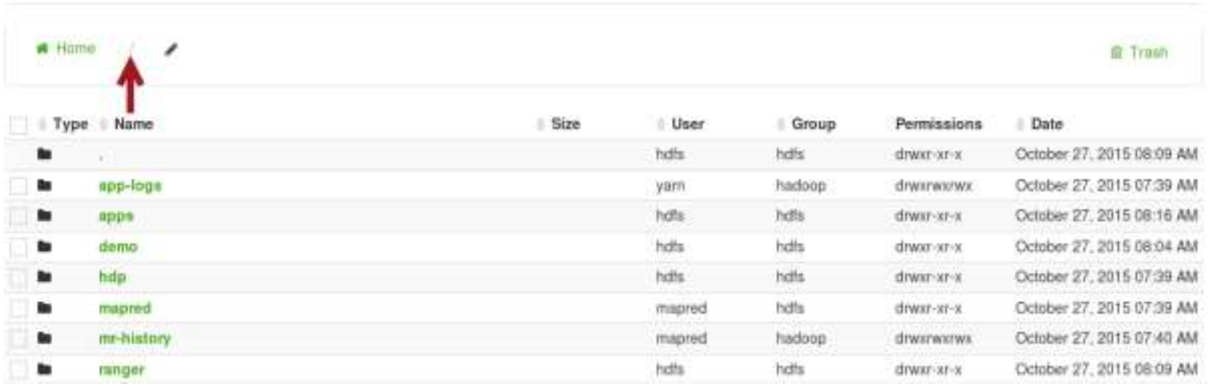
- Open a web browser on your VM, and navigate to bookmark tab and select HUE. This can be alternatively launched by specifying

- Enter username as ‘cloudera’ and password ‘cloudera’.

- Navigate to file browser

By default, the content of your HDFS home directory will be displayed. Locate the input directory created in previous step on the file browser.

- Click on leading slash to view the content of root directory

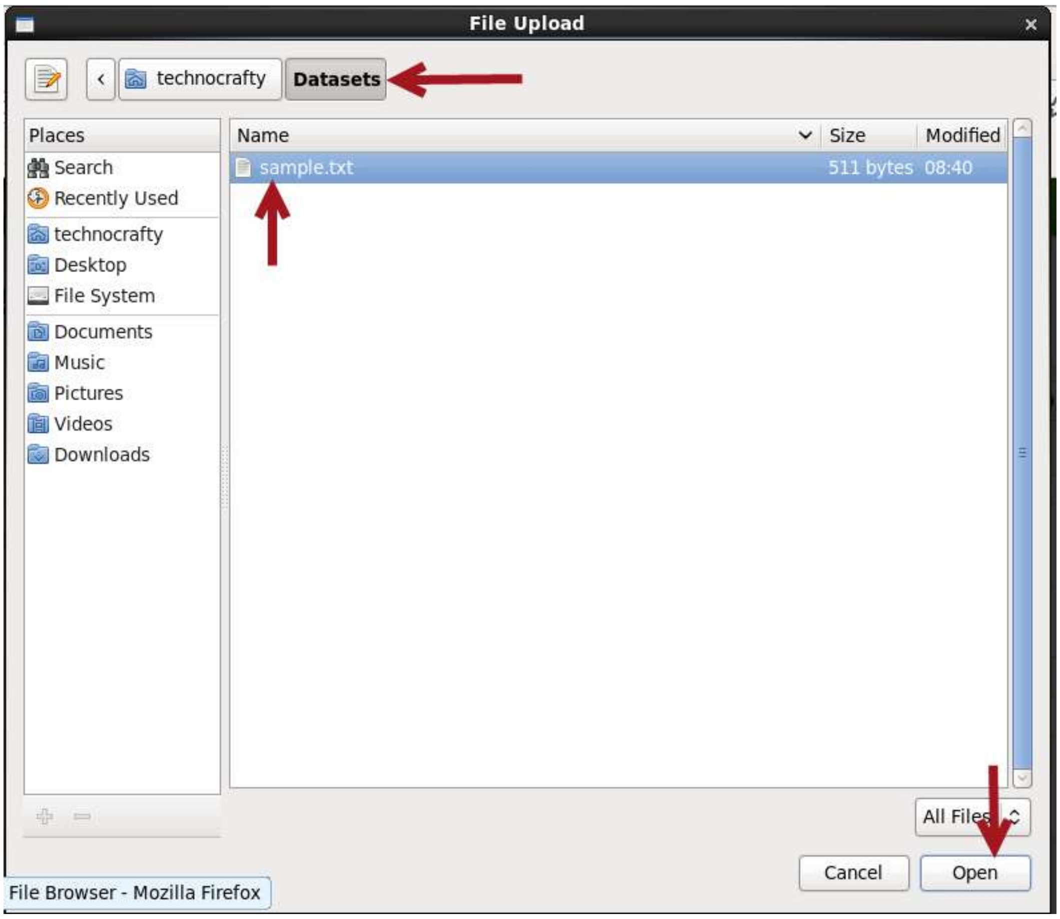


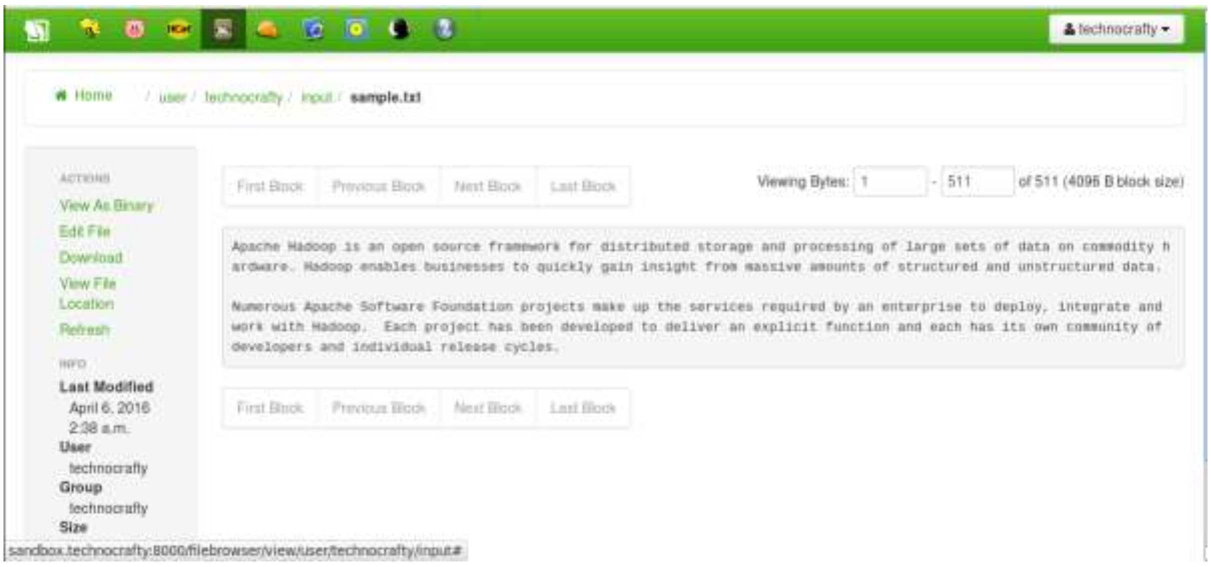
Type	Name	Size	User	Group	Permissions	Date
■	.		hdfs	hdfs	drwxr-xr-x	October 27, 2015 08:09 AM
■	app-logs		yarn	hadoop	drwxrwxrwx	October 27, 2015 07:39 AM
■	apps		hdfs	hdfs	drwxr-xr-x	October 27, 2015 08:16 AM
■	demo		hdfs	hdfs	drwxr-xr-x	October 27, 2015 08:04 AM
■	hdp		hdfs	hdfs	drwxr-xr-x	October 27, 2015 07:39 AM
■	mapred		mapred	hdfs	drwxr-xr-x	October 27, 2015 07:39 AM
■	mr-history		mapred	hadoop	drwxrwxrwx	October 27, 2015 07:40 AM
■	ranger		hdfs	hdfs	drwxr-xr-x	October 27, 2015 08:09 AM

- To perform operation on a file, select the file and several options will be enabled

- To upload the file, click on Upload button. From the drop down you can choose to upload a plain file or zipped file

- Click on Upload > Files > Select files





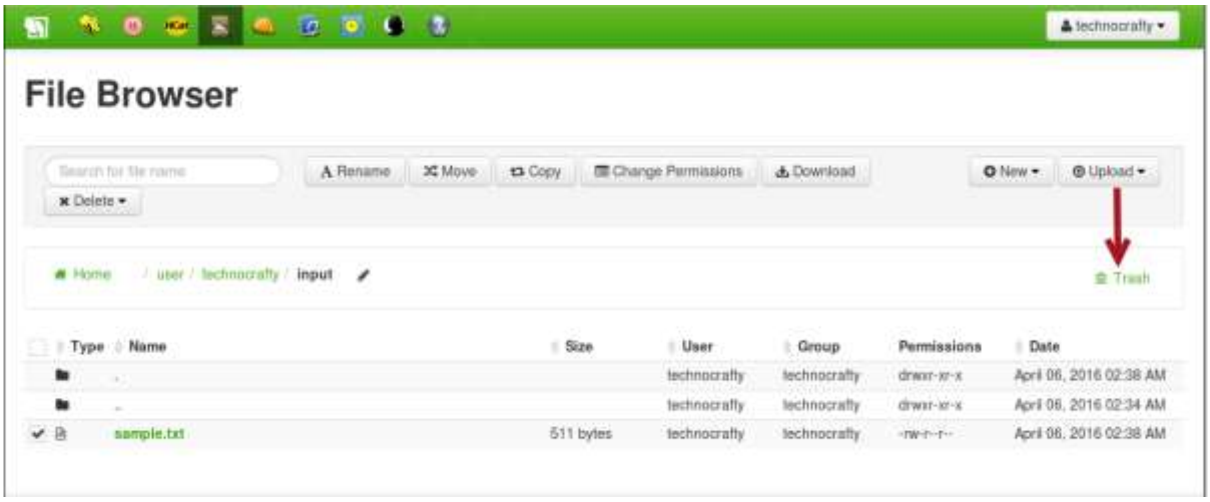
Last Modified
April 6, 2016
2:38 a.m.
User
technocrafty
Group
technocrafty
Size

Viewing Bytes: 1 - 511 of 511 (4096 B block size)

```
Apache Hadoop is an open source framework for distributed storage and processing of large sets of data on commodity h
ardware. Hadoop enables businesses to quickly gain insight from massive amounts of structured and unstructured data.

Numerous Apache Software Foundation projects make up the services required by an enterprise to deploy, integrate and
work with Hadoop.  Each project has been developed to deliver an explicit function and each has its own community of
developers and individual release cycles.
```

sandbox.technocrafty:8000/filebrowser/view/user/technocrafty/input#

➢ To remove the file, move it to Trash



	Type	Name	Size	User	Group	Permissions	Date
		.		technocrafty	technocrafty	drwxr-xr-x	April 06, 2016 02:38 AM
		..		technocrafty	technocrafty	drwxr-xr-x	April 06, 2016 02:34 AM
✓		sample.txt	511 bytes	technocrafty	technocrafty	-rw-r--r--	April 06, 2016 02:38 AM

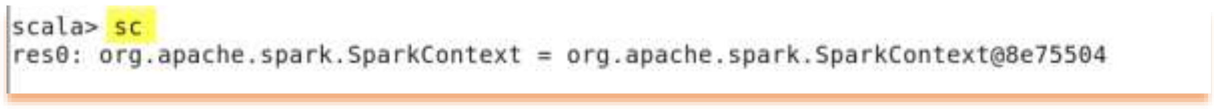
Exercise 3: Spark

Overview
In this lab, we will look at several transformations on RDD
Builds on
Previous labs for the transformations we'll use.
Run time
20-30 minutes

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

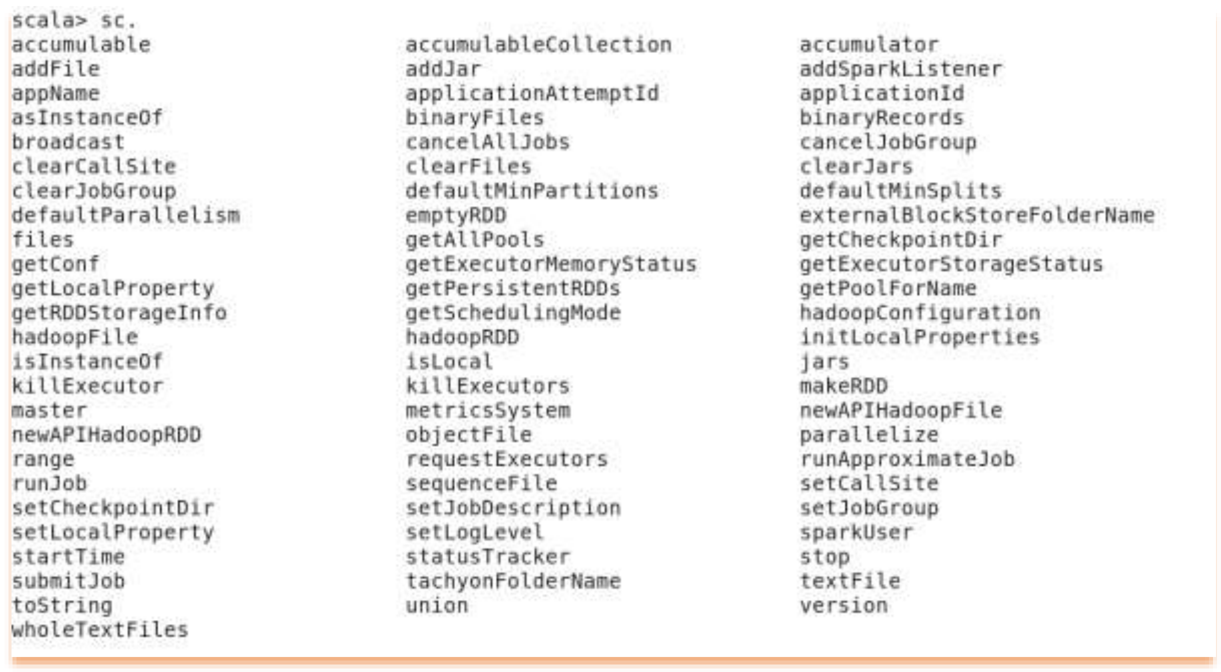
```
$ cd <path>/spark-2.4.4-bin-hadoop2.6

$ bin/spark-shell
```

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



```
scala> sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@8e75504
```

- 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                           jars
killExecutor                      killExecutors                     makeRDD
master                            metricsSystem                     newAPIHadoopFile
newAPIHadoopRDD                   objectFile                        parallelize
range                             requestExecutors                  runApproximateJob
runJob                            sequenceFile                      setCallSite
setCheckpointDir                  setJobDescription                 setJobGroup
setLocalProperty                  setLogLevel                       sparkUser
startTime                         statusTracker                     stop
submitJob                         tachyonFolderName                 textFile
toString                          union                             version
wholeTextFiles
```

1. Basic Commands and Operations

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

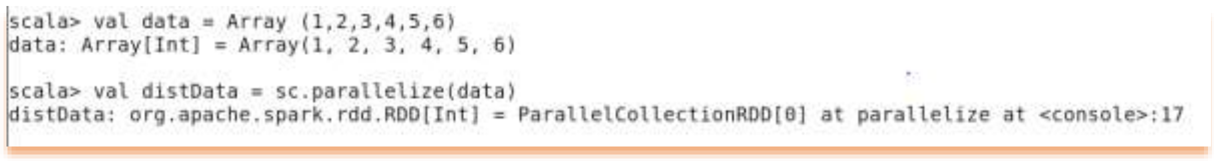
Two ways to create RDDs:

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

Parallelized collection:

To create a parallelized collection holding numbers 1 to 6

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

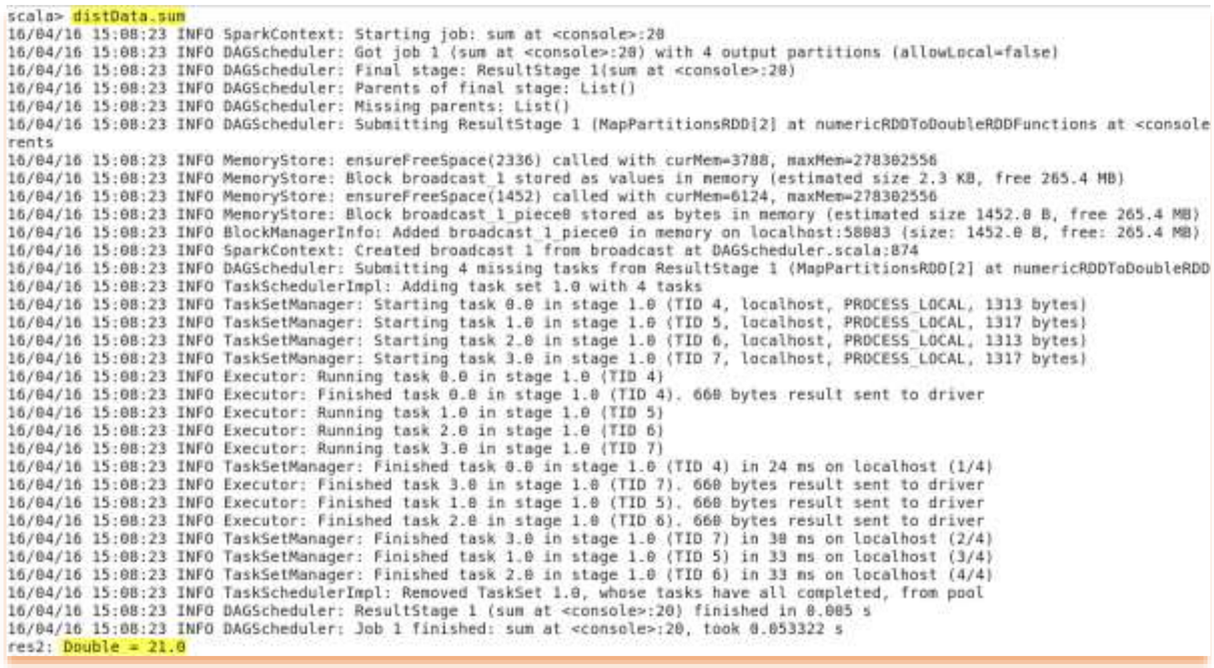


```
scala> val data = Array (1,2,3,4,5,6)
data: Array[Int] = Array(1, 2, 3, 4, 5, 6)

scala> val distData = sc.parallelize(data)
distData: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:17
```

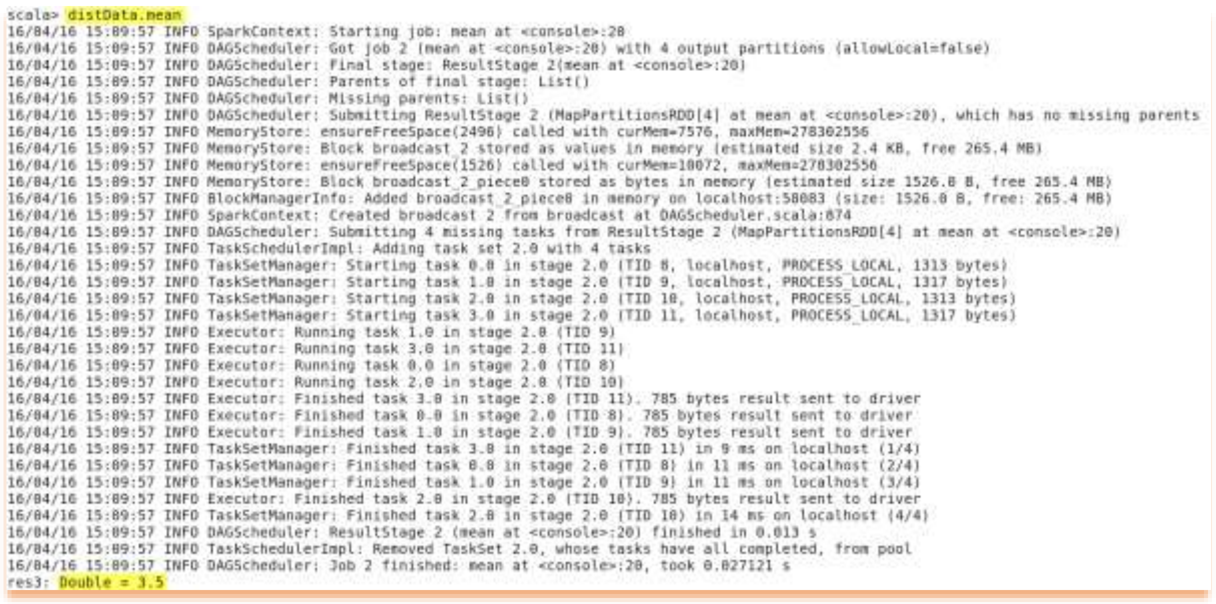
Once 'distData' dataset is created, we can perform operations such as:

- Finding the sum, mean & variance

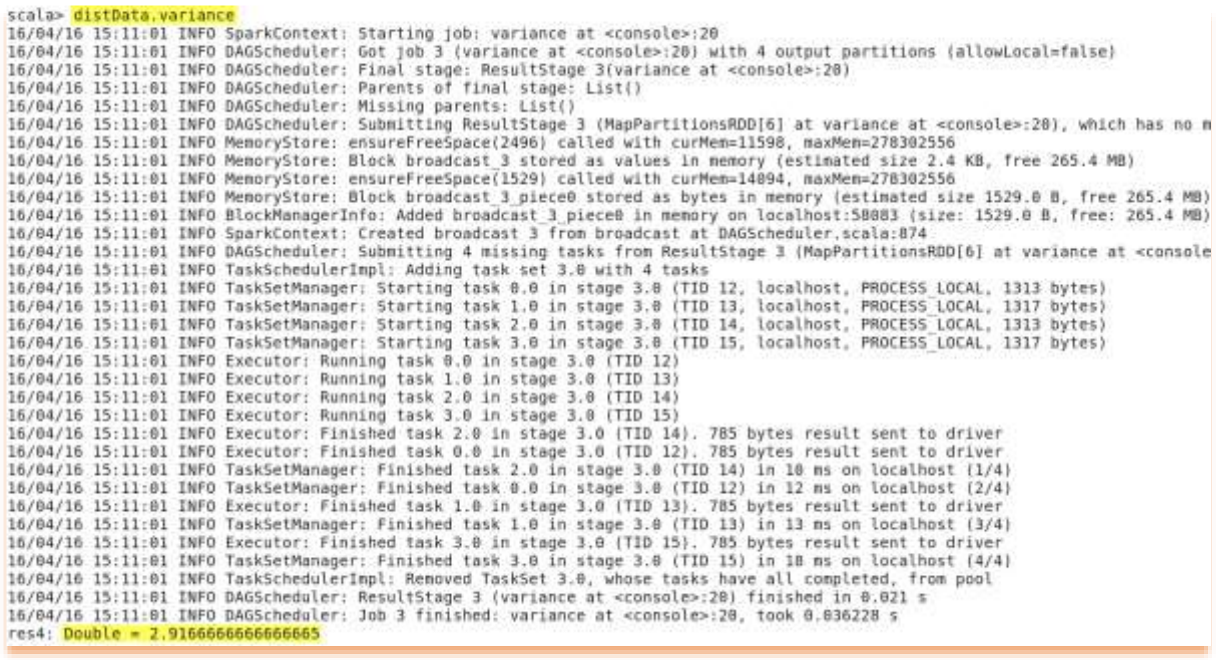
```
distData.sum
distData.mean
distData.variance
```



```
scala> distData.sum
16/04/16 15:08:23 INFO SparkContext: Starting job: sum at <console>:20
16/04/16 15:08:23 INFO DAGScheduler: Got job 1 (sum at <console>:20) with 4 output partitions (allowLocal=false)
16/04/16 15:08:23 INFO DAGScheduler: Final stage: ResultStage 1(sum at <console>:20)
16/04/16 15:08:23 INFO DAGScheduler: Parents of final stage: List()
16/04/16 15:08:23 INFO DAGScheduler: Missing parents: List()
16/04/16 15:08:23 INFO DAGScheduler: Submitting ResultStage 1 (MapPartitionsRDD[2] at numericRDDToDoubleRDDFunctions at <console
rents
16/04/16 15:08:23 INFO MemoryStore: ensureFreeSpace(2336) called with curMem=3788, maxMem=278302556
16/04/16 15:08:23 INFO MemoryStore: Block broadcast_1 stored as values in memory (estimated size 2.3 KB, free 265.4 MB)
16/04/16 15:08:23 INFO MemoryStore: ensureFreeSpace(1452) called with curMem=6124, maxMem=278302556
16/04/16 15:08:23 INFO MemoryStore: Block broadcast_1_piece0 stored as bytes in memory (estimated size 1452.0 B, free 265.4 MB)
16/04/16 15:08:23 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on localhost:58883 (size: 1452.0 B, free: 265.4 MB)
16/04/16 15:08:23 INFO SparkContext: Created broadcast 1 from broadcast at DAGScheduler.scala:874
16/04/16 15:08:23 INFO DAGScheduler: Submitting 4 missing tasks from ResultStage 1 (MapPartitionsRDD[2] at numericRDDToDoubleRDD
16/04/16 15:08:23 INFO TaskSchedulerImpl: Adding task set 1.0 with 4 tasks
16/04/16 15:08:23 INFO TaskSetManager: Starting task 0.0 in stage 1.0 (TID 4, localhost, PROCESS_LOCAL, 1313 bytes)
16/04/16 15:08:23 INFO TaskSetManager: Starting task 1.0 in stage 1.0 (TID 5, localhost, PROCESS_LOCAL, 1317 bytes)
16/04/16 15:08:23 INFO TaskSetManager: Starting task 2.0 in stage 1.0 (TID 6, localhost, PROCESS_LOCAL, 1313 bytes)
16/04/16 15:08:23 INFO TaskSetManager: Starting task 3.0 in stage 1.0 (TID 7, localhost, PROCESS_LOCAL, 1317 bytes)
16/04/16 15:08:23 INFO Executor: Running task 0.0 in stage 1.0 (TID 4)
16/04/16 15:08:23 INFO Executor: Finished task 0.0 in stage 1.0 (TID 4). 660 bytes result sent to driver
16/04/16 15:08:23 INFO Executor: Running task 1.0 in stage 1.0 (TID 5)
16/04/16 15:08:23 INFO Executor: Running task 2.0 in stage 1.0 (TID 6)
16/04/16 15:08:23 INFO Executor: Running task 3.0 in stage 1.0 (TID 7)
16/04/16 15:08:23 INFO TaskSetManager: Finished task 0.0 in stage 1.0 (TID 4) in 24 ms on localhost (1/4)
16/04/16 15:08:23 INFO Executor: Finished task 3.0 in stage 1.0 (TID 7). 660 bytes result sent to driver
16/04/16 15:08:23 INFO Executor: Finished task 1.0 in stage 1.0 (TID 5). 660 bytes result sent to driver
16/04/16 15:08:23 INFO Executor: Finished task 2.0 in stage 1.0 (TID 6). 660 bytes result sent to driver
16/04/16 15:08:23 INFO TaskSetManager: Finished task 3.0 in stage 1.0 (TID 7) in 38 ms on localhost (2/4)
16/04/16 15:08:23 INFO TaskSetManager: Finished task 1.0 in stage 1.0 (TID 5) in 33 ms on localhost (3/4)
16/04/16 15:08:23 INFO TaskSetManager: Finished task 2.0 in stage 1.0 (TID 6) in 33 ms on localhost (4/4)
16/04/16 15:08:23 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
16/04/16 15:08:23 INFO DAGScheduler: ResultStage 1 (sum at <console>:20) finished in 0.085 s
16/04/16 15:08:23 INFO DAGScheduler: Job 1 finished: sum at <console>:20, took 0.053322 s
res2: Double = 21.0
```



```
scala> distData.mean
16/04/16 15:09:57 INFO SparkContext: Starting job: mean at <console>:20
16/04/16 15:09:57 INFO DAGScheduler: Got job 2 (mean at <console>:20) with 4 output partitions (allowLocal=false)
16/04/16 15:09:57 INFO DAGScheduler: Final stage: ResultStage 2(mean at <console>:20)
16/04/16 15:09:57 INFO DAGScheduler: Parents of final stage: List()
16/04/16 15:09:57 INFO DAGScheduler: Missing parents: List()
16/04/16 15:09:57 INFO DAGScheduler: Submitting ResultStage 2 (MapPartitionsRDD[4] at mean at <console>:20), which has no missing parents
16/04/16 15:09:57 INFO MemoryStore: ensureFreeSpace(2496) called with curMem=7576, maxMem=278302556
16/04/16 15:09:57 INFO MemoryStore: Block broadcast_2 stored as values in memory (estimated size 2.4 KB, free 265.4 MB)
16/04/16 15:09:57 INFO MemoryStore: ensureFreeSpace(1526) called with curMem=10072, maxMem=278302556
16/04/16 15:09:57 INFO MemoryStore: Block broadcast_2_piece0 stored as bytes in memory (estimated size 1526.0 B, free 265.4 MB)
16/04/16 15:09:57 INFO BlockManagerInfo: Added broadcast_2_piece0 in memory on localhost:58083 (size: 1526.0 B, free: 265.4 MB)
16/04/16 15:09:57 INFO SparkContext: Created broadcast 2 from broadcast at DAGScheduler.scala:874
16/04/16 15:09:57 INFO DAGScheduler: Submitting 4 missing tasks from ResultStage 2 (MapPartitionsRDD[4] at mean at <console>:20)
16/04/16 15:09:57 INFO TaskSchedulerImpl: Adding task set 2.0 with 4 tasks
16/04/16 15:09:57 INFO TaskSetManager: Starting task 0.0 in stage 2.0 (TID 8, localhost, PROCESS_LOCAL, 1313 bytes)
16/04/16 15:09:57 INFO TaskSetManager: Starting task 1.0 in stage 2.0 (TID 9, localhost, PROCESS_LOCAL, 1317 bytes)
16/04/16 15:09:57 INFO TaskSetManager: Starting task 2.0 in stage 2.0 (TID 10, localhost, PROCESS_LOCAL, 1313 bytes)
16/04/16 15:09:57 INFO TaskSetManager: Starting task 3.0 in stage 2.0 (TID 11, localhost, PROCESS_LOCAL, 1317 bytes)
16/04/16 15:09:57 INFO Executor: Running task 1.0 in stage 2.0 (TID 9)
16/04/16 15:09:57 INFO Executor: Running task 3.0 in stage 2.0 (TID 11)
16/04/16 15:09:57 INFO Executor: Running task 0.0 in stage 2.0 (TID 8)
16/04/16 15:09:57 INFO Executor: Running task 2.0 in stage 2.0 (TID 10)
16/04/16 15:09:57 INFO Executor: Finished task 3.0 in stage 2.0 (TID 11). 785 bytes result sent to driver
16/04/16 15:09:57 INFO Executor: Finished task 0.0 in stage 2.0 (TID 8). 785 bytes result sent to driver
16/04/16 15:09:57 INFO Executor: Finished task 1.0 in stage 2.0 (TID 9). 785 bytes result sent to driver
16/04/16 15:09:57 INFO TaskSetManager: Finished task 3.0 in stage 2.0 (TID 11) in 9 ms on localhost (1/4)
16/04/16 15:09:57 INFO TaskSetManager: Finished task 0.0 in stage 2.0 (TID 8) in 11 ms on localhost (2/4)
16/04/16 15:09:57 INFO TaskSetManager: Finished task 1.0 in stage 2.0 (TID 9) in 11 ms on localhost (3/4)
16/04/16 15:09:57 INFO Executor: Finished task 2.0 in stage 2.0 (TID 10). 785 bytes result sent to driver
16/04/16 15:09:57 INFO TaskSetManager: Finished task 2.0 in stage 2.0 (TID 10) in 14 ms on localhost (4/4)
16/04/16 15:09:57 INFO DAGScheduler: ResultStage 2 (mean at <console>:20) finished in 0.013 s
16/04/16 15:09:57 INFO TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all completed, from pool
16/04/16 15:09:57 INFO DAGScheduler: Job 2 finished: mean at <console>:20, took 0.027121 s
res3: Double = 3.5
```



```
scala> distData.variance
16/04/16 15:11:01 INFO SparkContext: Starting job: variance at <console>:20
16/04/16 15:11:01 INFO DAGScheduler: Got job 3 (variance at <console>:20) with 4 output partitions (allowLocal=false)
16/04/16 15:11:01 INFO DAGScheduler: Final stage: ResultStage 3(variance at <console>:20)
16/04/16 15:11:01 INFO DAGScheduler: Parents of final stage: List()
16/04/16 15:11:01 INFO DAGScheduler: Missing parents: List()
16/04/16 15:11:01 INFO DAGScheduler: Submitting ResultStage 3 (MapPartitionsRDD[6] at variance at <console>:20), which has no m
16/04/16 15:11:01 INFO MemoryStore: ensureFreeSpace(2496) called with curMem=11598, maxMem=278302556
16/04/16 15:11:01 INFO MemoryStore: Block broadcast_3 stored as values in memory (estimated size 2.4 KB, free 265.4 MB)
16/04/16 15:11:01 INFO MemoryStore: ensureFreeSpace(1529) called with curMem=14094, maxMem=278302556
16/04/16 15:11:01 INFO MemoryStore: Block broadcast_3_piece0 stored as bytes in memory (estimated size 1529.0 B, free 265.4 MB)
16/04/16 15:11:01 INFO BlockManagerInfo: Added broadcast_3_piece0 in memory on localhost:58083 (size: 1529.0 B, free: 265.4 MB)
16/04/16 15:11:01 INFO SparkContext: Created broadcast 3 from broadcast at DAGScheduler.scala:874
16/04/16 15:11:01 INFO DAGScheduler: Submitting 4 missing tasks from ResultStage 3 (MapPartitionsRDD[6] at variance at <console
16/04/16 15:11:01 INFO TaskSchedulerImpl: Adding task set 3.0 with 4 tasks
16/04/16 15:11:01 INFO TaskSetManager: Starting task 0.0 in stage 3.0 (TID 12, localhost, PROCESS_LOCAL, 1313 bytes)
16/04/16 15:11:01 INFO TaskSetManager: Starting task 1.0 in stage 3.0 (TID 13, localhost, PROCESS_LOCAL, 1317 bytes)
16/04/16 15:11:01 INFO TaskSetManager: Starting task 2.0 in stage 3.0 (TID 14, localhost, PROCESS_LOCAL, 1313 bytes)
16/04/16 15:11:01 INFO TaskSetManager: Starting task 3.0 in stage 3.0 (TID 15, localhost, PROCESS_LOCAL, 1317 bytes)
16/04/16 15:11:01 INFO Executor: Running task 0.0 in stage 3.0 (TID 12)
16/04/16 15:11:01 INFO Executor: Running task 1.0 in stage 3.0 (TID 13)
16/04/16 15:11:01 INFO Executor: Running task 2.0 in stage 3.0 (TID 14)
16/04/16 15:11:01 INFO Executor: Running task 3.0 in stage 3.0 (TID 15)
16/04/16 15:11:01 INFO Executor: Finished task 2.0 in stage 3.0 (TID 14). 785 bytes result sent to driver
16/04/16 15:11:01 INFO Executor: Finished task 0.0 in stage 3.0 (TID 12). 785 bytes result sent to driver
16/04/16 15:11:01 INFO TaskSetManager: Finished task 2.0 in stage 3.0 (TID 14) in 10 ms on localhost (1/4)
16/04/16 15:11:01 INFO TaskSetManager: Finished task 0.0 in stage 3.0 (TID 12) in 12 ms on localhost (2/4)
16/04/16 15:11:01 INFO Executor: Finished task 1.0 in stage 3.0 (TID 13). 785 bytes result sent to driver
16/04/16 15:11:01 INFO TaskSetManager: Finished task 1.0 in stage 3.0 (TID 13) in 13 ms on localhost (3/4)
16/04/16 15:11:01 INFO Executor: Finished task 3.0 in stage 3.0 (TID 15). 785 bytes result sent to driver
16/04/16 15:11:01 INFO TaskSetManager: Finished task 3.0 in stage 3.0 (TID 15) in 10 ms on localhost (4/4)
16/04/16 15:11:01 INFO TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have all completed, from pool
16/04/16 15:11:01 INFO DAGScheduler: ResultStage 3 (variance at <console>:20) finished in 0.021 s
16/04/16 15:11:01 INFO DAGScheduler: Job 3 finished: variance at <console>:20, took 0.036228 s
res4: Double = 2.9166666666666665
```

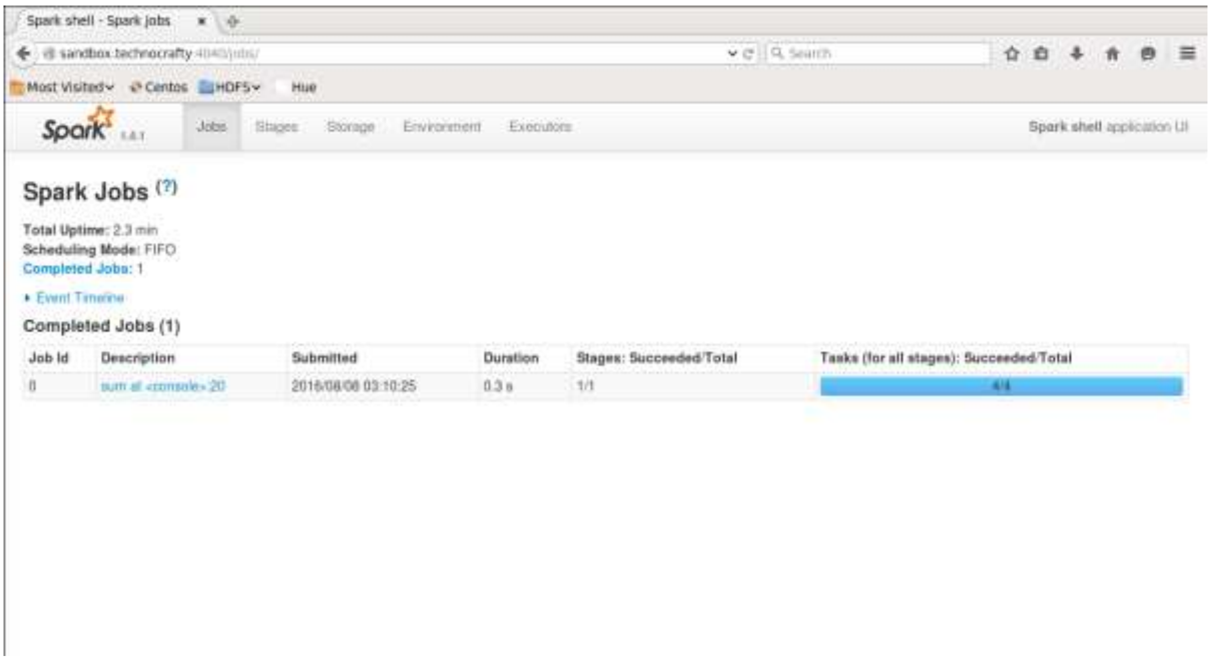
Note

Spark sets the number of partition (i.e.to cut the dataset into) automatically based on your cluster, but this can be set manually by passing second parameter to *parallelize* (e.g. sc.parallelize(data,10))

Spark UI

Spark UI can be accessed on port <localhost:4040>

Job is created once you perform an action.



Spark Jobs (?)

Total Uptime: 2.3 min
Scheduling Mode: FIFO
Completed Jobs: 1

▸ Event Timeline

Completed Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	sum at <console>:20	2016/08/08 03:10:25	0.3 s	1/1	4/4

External Datasets:

Spark can create distributed datasets from any storage system supported by Hadoop, Spark supports text files, SequenceFiles and any other Hadoop InputFormat.

To load a file from your Local Filesystem

Example2

```
val textFile = sc.textFile("<path>/Datasets/Employee.txt")
```

Remember!

While using local filesystem, the file must be accessible at the same path on worker nodes.

Operations on RDD will be discussed in next section.

Note

Spark's file-based input methods supports directories, compressed files and wildcards as well. i.e. textFile("/my/directory"), textFile("/my/directory/*.txt"), and textFile("/my/directory/*.gz")

2. Operations on RDD

RDDs supports two types of operations:

- **Transformations**

Creates a new dataset from an existing one

- **Actions**

Returns a value to the driver program after running a computation on the dataset.

Example: map is a transformation that passes each dataset element through a function and return a new RDD, whereas reduce is an action that aggregates all the elements of RDD using function and returns final result.

Note

All transformations in Spark are lazy i.e. transformations are only computed when an action requires a result to be returned.

textFile dataset was already created in previous step, lets perform below operations

- counting the occurrence of lines having a particular word in it

Example: filter using scala shell

```
val textFile = sc.textFile("<path>/Datasets/Employee.txt")
textFile.count() //number of items in this RDD
textFile.first() //First  item in this RDD
val linesWithHadoop = textFile.filter(line => line.contains("7839"))
linesWithHadoop.count()
```

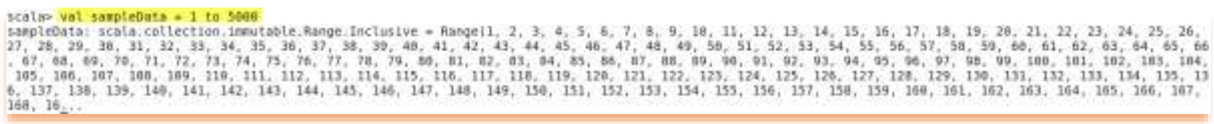
- Add up the sizes of all the lines

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

val totalLength = lineLength.reduce((a, b) => a + b)
```

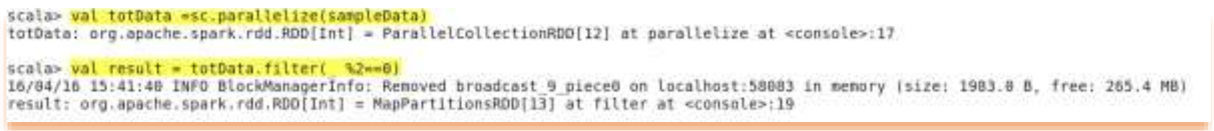
- Line with maximum words

```
Example:

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

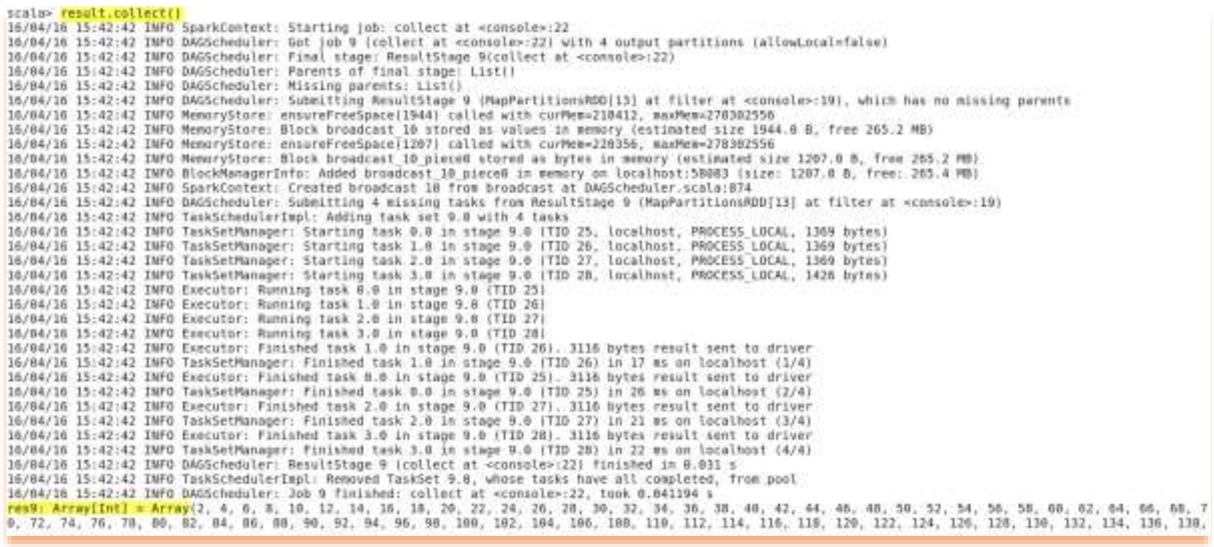
```
Example4: filter

val sampleData = 1 to 5000
val totData = sc.parallelize(sampleData)
val result = totData.filter(_ %2==0)
result.collect()


with number of partition = 2

val totDataPar = sc.parallelize(sampleData,2)
val resultPar  = totDataPar.filter(_%2==0)
resultPar.collect()
```



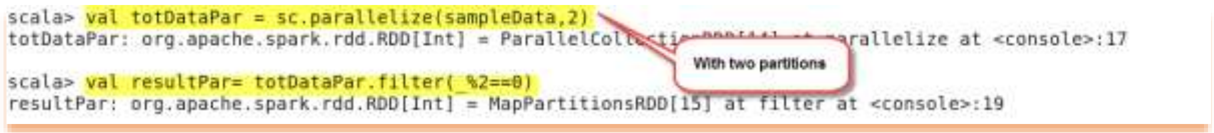
```
scala> val sampleData = 1 to 5000
sampleData: scala.collection.immutable.Range.Inclusive = Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66
, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104,
 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 13
6, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167,
168, 16...
```



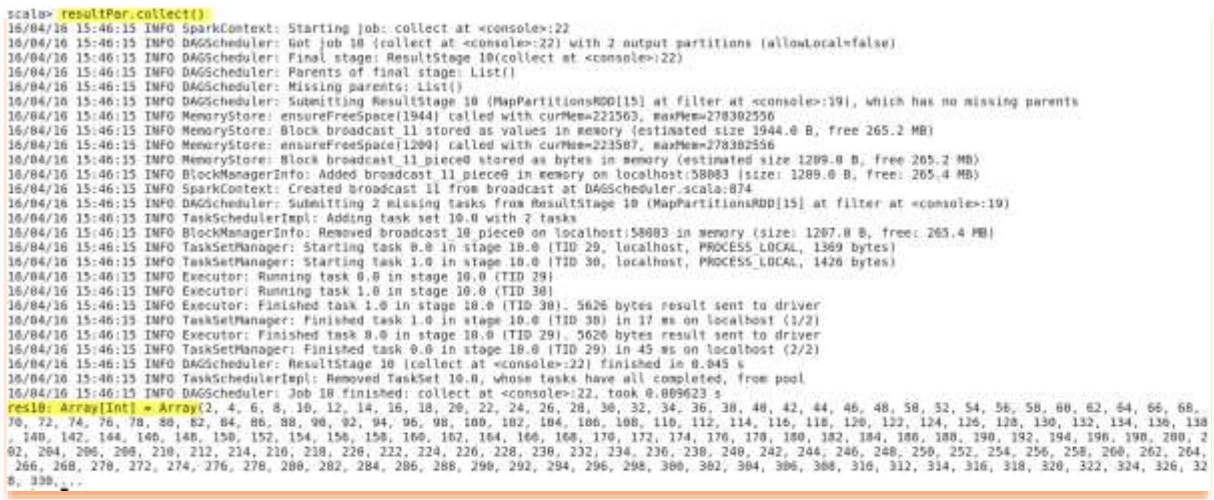
```
scala> val totData =sc.parallelize(sampleData)
totData: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[12] at parallelize at <console>:17

scala> val result = totData.filter(_ %2==0)
16/04/16 15:41:40 INFO BlockManagerInfo: Removed broadcast_9_piece0 on localhost:58083 in memory (size: 1983.0 B, free: 265.4 MB)
result: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[13] at filter at <console>:19
```



```
scala> result.collect()
16/04/16 15:42:42 INFO SparkContext: Starting job: collect at <console>:22
16/04/16 15:42:42 INFO DAGScheduler: Got job 9 (collect at <console>:22) with 4 output partitions (allowLocal=false)
16/04/16 15:42:42 INFO DAGScheduler: Final stage: ResultStage 9(collect at <console>:22)
16/04/16 15:42:42 INFO DAGScheduler: Parents of final stage: List()
16/04/16 15:42:42 INFO DAGScheduler: Missing parents: List()
16/04/16 15:42:42 INFO DAGScheduler: Submitting ResultStage 9 (MapPartitionsRDD[13] at filter at <console>:19), which has no missing parents
16/04/16 15:42:42 INFO MemoryStore: ensureFreeSpace(1944) called with curMem=218412, maxMem=278302556
16/04/16 15:42:42 INFO MemoryStore: Block broadcast_10 stored as values in memory (estimated size 1944.0 B, free 265.2 MB)
16/04/16 15:42:42 INFO MemoryStore: ensureFreeSpace(1207) called with curMem=220356, maxMem=278302556
16/04/16 15:42:42 INFO MemoryStore: Block broadcast_10_piece0 stored as bytes in memory (estimated size 1207.0 B, free 265.2 MB)
16/04/16 15:42:42 INFO BlockManagerInfo: Added broadcast_10_piece0 in memory on localhost:58083 (size: 1207.0 B, free: 265.4 MB)
16/04/16 15:42:42 INFO SparkContext: Created broadcast 10 from broadcast at DAGScheduler.scala:874
16/04/16 15:42:42 INFO DAGScheduler: Submitting 4 missing tasks from ResultStage 9 (MapPartitionsRDD[13] at filter at <console>:19)
16/04/16 15:42:42 INFO TaskSchedulerImpl: Adding task set 9.0 with 4 tasks
16/04/16 15:42:42 INFO TaskSetManager: Starting task 0.0 in stage 9.0 (TID 25, localhost, PROCESS_LOCAL, 1369 bytes)
16/04/16 15:42:42 INFO TaskSetManager: Starting task 1.0 in stage 9.0 (TID 26, localhost, PROCESS_LOCAL, 1369 bytes)
16/04/16 15:42:42 INFO TaskSetManager: Starting task 2.0 in stage 9.0 (TID 27, localhost, PROCESS_LOCAL, 1369 bytes)
16/04/16 15:42:42 INFO TaskSetManager: Starting task 3.0 in stage 9.0 (TID 28, localhost, PROCESS_LOCAL, 1426 bytes)
16/04/16 15:42:42 INFO Executor: Running task 0.0 in stage 9.0 (TID 25)
16/04/16 15:42:42 INFO Executor: Running task 1.0 in stage 9.0 (TID 26)
16/04/16 15:42:42 INFO Executor: Running task 2.0 in stage 9.0 (TID 27)
16/04/16 15:42:42 INFO Executor: Running task 3.0 in stage 9.0 (TID 28)
16/04/16 15:42:42 INFO Executor: Finished task 1.0 in stage 9.0 (TID 26). 3116 bytes result sent to driver
16/04/16 15:42:42 INFO TaskSetManager: Finished task 1.0 in stage 9.0 (TID 26) in 17 ms on localhost (1/4)
16/04/16 15:42:42 INFO Executor: Finished task 0.0 in stage 9.0 (TID 25). 3116 bytes result sent to driver
16/04/16 15:42:42 INFO TaskSetManager: Finished task 0.0 in stage 9.0 (TID 25) in 26 ms on localhost (2/4)
16/04/16 15:42:42 INFO Executor: Finished task 2.0 in stage 9.0 (TID 27). 3116 bytes result sent to driver
16/04/16 15:42:42 INFO TaskSetManager: Finished task 2.0 in stage 9.0 (TID 27) in 21 ms on localhost (3/4)
16/04/16 15:42:42 INFO Executor: Finished task 3.0 in stage 9.0 (TID 28). 3116 bytes result sent to driver
16/04/16 15:42:42 INFO TaskSetManager: Finished task 3.0 in stage 9.0 (TID 28) in 22 ms on localhost (4/4)
16/04/16 15:42:42 INFO DAGScheduler: ResultStage 9 (collect at <console>:22) finished in 0.031 s
16/04/16 15:42:42 INFO TaskSchedulerImpl: Removed TaskSet 9.0, whose tasks have all completed, from pool
16/04/16 15:42:42 INFO DAGScheduler: Job 9 finished: collect at <console>:22, took 0.041194 s
res9: Array[Int] = Array(2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 7
0, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138,
```



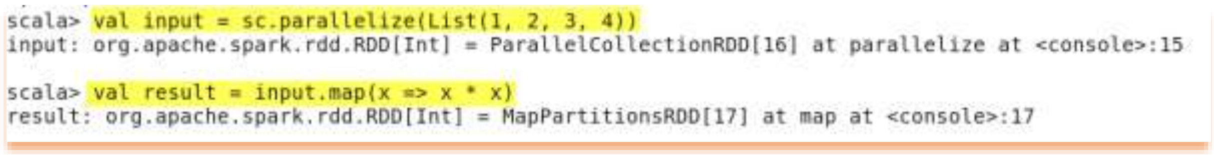
```
scala> val totDataPar = sc.parallelize(sampleData,2)
totDataPar: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[14] at parallelize at <console>:17

scala> val resultPar= totDataPar.filter(_%2==0)
resultPar: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[15] at filter at <console>:19
```



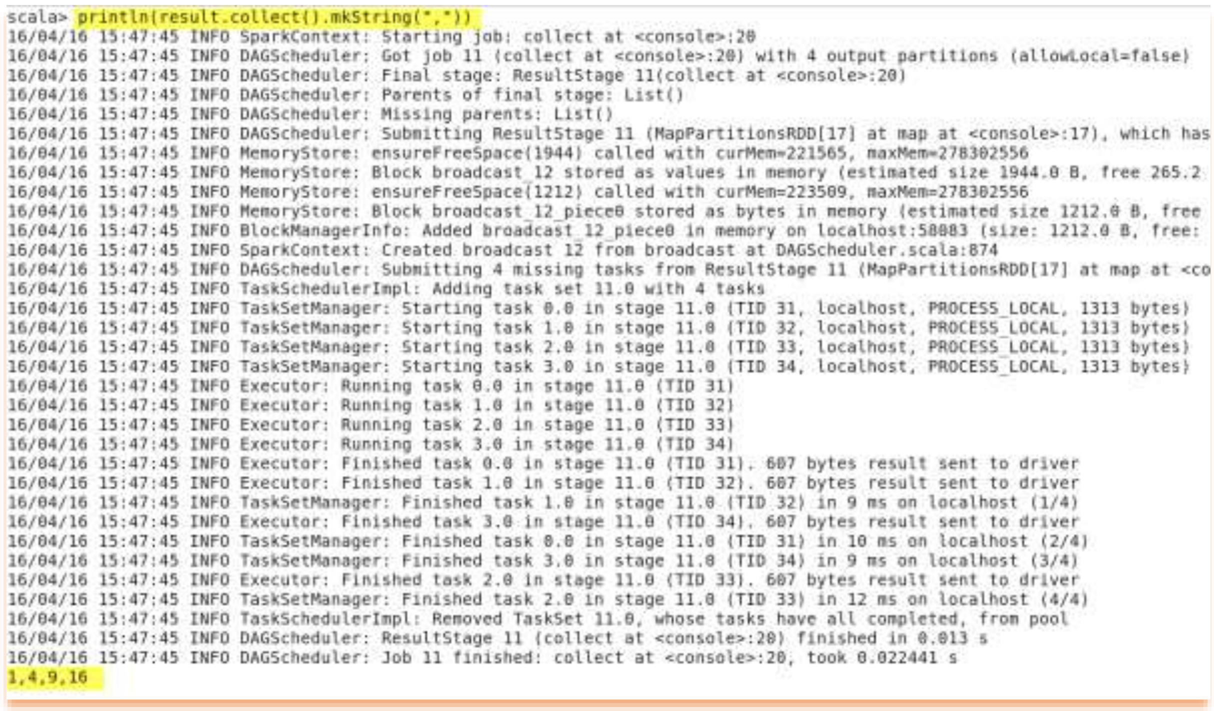
```
scala> resultPar.collect()
16/04/16 15:46:15 INFO SparkContext: Starting job: collect at <console>:22
16/04/16 15:46:15 INFO DAGScheduler: Got job 10 (collect at <console>:22) with 2 output partitions (allowLocal=false)
16/04/16 15:46:15 INFO DAGScheduler: Final stage: ResultStage 10(collect at <console>:22)
16/04/16 15:46:15 INFO DAGScheduler: Parents of final stage: List()
16/04/16 15:46:15 INFO DAGScheduler: Missing parents: List()
16/04/16 15:46:15 INFO DAGScheduler: Submitting ResultStage 10 (MapPartitionsRDD[15] at filter at <console>:19), which has no missing parents
16/04/16 15:46:15 INFO MemoryStore: ensureFreeSpace(1944) called with curMem=221563, maxMem=278302556
16/04/16 15:46:15 INFO MemoryStore: Block broadcast_11 stored as values in memory (estimated size 1944.0 B, free 265.2 MB)
16/04/16 15:46:15 INFO MemoryStore: ensureFreeSpace(1209) called with curMem=223507, maxMem=278302556
16/04/16 15:46:15 INFO MemoryStore: Block broadcast_11_piece0 stored as bytes in memory (estimated size 1209.0 B, free 265.2 MB)
16/04/16 15:46:15 INFO BlockManagerInfo: Added broadcast_11_piece0 in memory on localhost:58083 (size: 1209.0 B, free: 265.4 MB)
16/04/16 15:46:15 INFO SparkContext: Created broadcast 11 from broadcast at DAGScheduler.scala:874
16/04/16 15:46:15 INFO DAGScheduler: Submitting 2 missing tasks from ResultStage 10 (MapPartitionsRDD[15] at filter at <console>:19)
16/04/16 15:46:15 INFO TaskSchedulerImpl: Adding task set 10.0 with 2 tasks
16/04/16 15:46:15 INFO BlockManagerInfo: Removed broadcast_10_piece0 on localhost:58083 in memory (size: 1207.0 B, free: 265.4 MB)
16/04/16 15:46:15 INFO TaskSetManager: Starting task 0.0 in stage 10.0 (TID 29, localhost, PROCESS_LOCAL, 1369 bytes)
16/04/16 15:46:15 INFO TaskSetManager: Starting task 1.0 in stage 10.0 (TID 30, localhost, PROCESS_LOCAL, 1426 bytes)
16/04/16 15:46:15 INFO Executor: Running task 0.0 in stage 10.0 (TID 29)
16/04/16 15:46:15 INFO Executor: Running task 1.0 in stage 10.0 (TID 30)
16/04/16 15:46:15 INFO Executor: Finished task 1.0 in stage 10.0 (TID 30). 5626 bytes result sent to driver
16/04/16 15:46:15 INFO TaskSetManager: Finished task 1.0 in stage 10.0 (TID 30) in 17 ms on localhost (1/2)
16/04/16 15:46:15 INFO Executor: Finished task 0.0 in stage 10.0 (TID 29). 5626 bytes result sent to driver
16/04/16 15:46:15 INFO TaskSetManager: Finished task 0.0 in stage 10.0 (TID 29) in 45 ms on localhost (2/2)
16/04/16 15:46:15 INFO DAGScheduler: ResultStage 10 (collect at <console>:22) finished in 0.045 s
16/04/16 15:46:15 INFO TaskSchedulerImpl: Removed TaskSet 10.0, whose tasks have all completed, from pool
16/04/16 15:46:15 INFO DAGScheduler: Job 10 finished: collect at <console>:22, took 0.089623 s
res10: Array[Int] = Array(2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68,
70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138
, 140, 142, 144, 146, 148, 150, 152, 154, 156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 178, 180, 182, 184, 186, 188, 190, 192, 194, 196, 198, 200, 2
02, 204, 206, 208, 210, 212, 214, 216, 218, 220, 222, 224, 226, 228, 230, 232, 234, 236, 238, 240, 242, 244, 246, 248, 250, 252, 254, 256, 258, 260, 262, 264,
 266, 268, 270, 272, 274, 276, 278, 280, 282, 284, 286, 288, 290, 292, 294, 296, 298, 300, 302, 304, 306, 308, 310, 312, 314, 316, 318, 320, 322, 324, 326, 32
8, 330,...
```

Example: Scala example of map

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

val result = input.map(x => x * x)

println(result.collect().mkString(","))
```



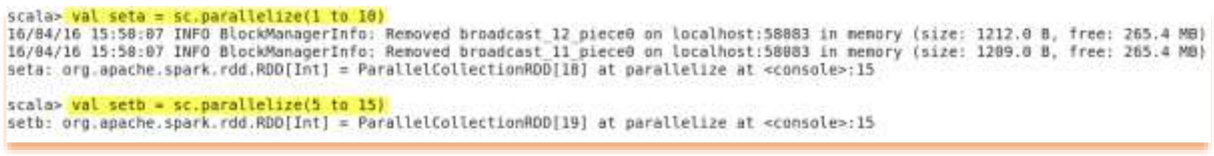
```
scala> val input = sc.parallelize(List(1, 2, 3, 4))
input: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[16] at parallelize at <console>:15

scala> val result = input.map(x => x * x)
result: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[17] at map at <console>:17
```



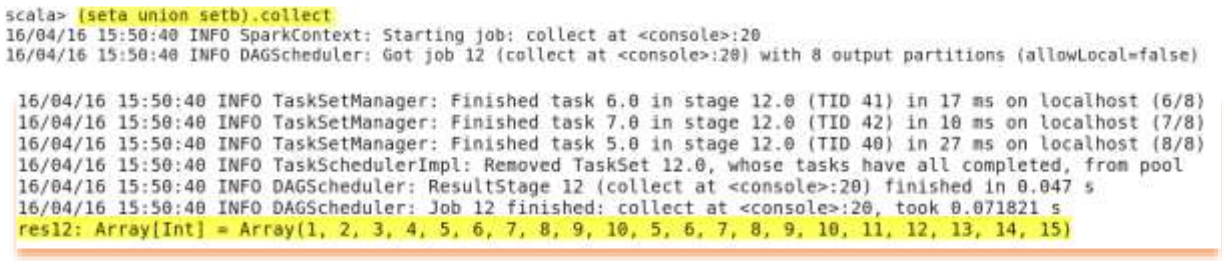
```
scala> println(result.collect().mkString(","))
16/04/16 15:47:45 INFO SparkContext: Starting job: collect at <console>:20
16/04/16 15:47:45 INFO DAGScheduler: Got job 11 (collect at <console>:20) with 4 output partitions (allowLocal=false)
16/04/16 15:47:45 INFO DAGScheduler: Final stage: ResultStage 11(collect at <console>:20)
16/04/16 15:47:45 INFO DAGScheduler: Parents of final stage: List()
16/04/16 15:47:45 INFO DAGScheduler: Missing parents: List()
16/04/16 15:47:45 INFO DAGScheduler: Submitting ResultStage 11 (MapPartitionsRDD[17] at map at <console>:17), which has
16/04/16 15:47:45 INFO MemoryStore: ensureFreeSpace(1944) called with curMem=221565, maxMem=278302556
16/04/16 15:47:45 INFO MemoryStore: Block broadcast_12 stored as values in memory (estimated size 1944.0 B, free 265.2
16/04/16 15:47:45 INFO MemoryStore: ensureFreeSpace(1212) called with curMem=223509, maxMem=278302556
16/04/16 15:47:45 INFO MemoryStore: Block broadcast_12_piece0 stored as bytes in memory (estimated size 1212.0 B, free
16/04/16 15:47:45 INFO BlockManagerInfo: Added broadcast_12_piece0 in memory on localhost:58883 (size: 1212.0 B, free:
16/04/16 15:47:45 INFO SparkContext: Created broadcast 12 from broadcast at DAGScheduler.scala:874
16/04/16 15:47:45 INFO DAGScheduler: Submitting 4 missing tasks from ResultStage 11 (MapPartitionsRDD[17] at map at <co
16/04/16 15:47:45 INFO TaskSchedulerImpl: Adding task set 11.0 with 4 tasks
16/04/16 15:47:45 INFO TaskSetManager: Starting task 0.0 in stage 11.0 (TID 31, localhost, PROCESS_LOCAL, 1313 bytes)
16/04/16 15:47:45 INFO TaskSetManager: Starting task 1.0 in stage 11.0 (TID 32, localhost, PROCESS_LOCAL, 1313 bytes)
16/04/16 15:47:45 INFO TaskSetManager: Starting task 2.0 in stage 11.0 (TID 33, localhost, PROCESS_LOCAL, 1313 bytes)
16/04/16 15:47:45 INFO TaskSetManager: Starting task 3.0 in stage 11.0 (TID 34, localhost, PROCESS_LOCAL, 1313 bytes)
16/04/16 15:47:45 INFO Executor: Running task 0.0 in stage 11.0 (TID 31)
16/04/16 15:47:45 INFO Executor: Running task 1.0 in stage 11.0 (TID 32)
16/04/16 15:47:45 INFO Executor: Running task 2.0 in stage 11.0 (TID 33)
16/04/16 15:47:45 INFO Executor: Running task 3.0 in stage 11.0 (TID 34)
16/04/16 15:47:45 INFO Executor: Finished task 0.0 in stage 11.0 (TID 31). 607 bytes result sent to driver
16/04/16 15:47:45 INFO Executor: Finished task 1.0 in stage 11.0 (TID 32). 607 bytes result sent to driver
16/04/16 15:47:45 INFO TaskSetManager: Finished task 1.0 in stage 11.0 (TID 32) in 9 ms on localhost (1/4)
16/04/16 15:47:45 INFO Executor: Finished task 3.0 in stage 11.0 (TID 34). 607 bytes result sent to driver
16/04/16 15:47:45 INFO TaskSetManager: Finished task 0.0 in stage 11.0 (TID 31) in 10 ms on localhost (2/4)
16/04/16 15:47:45 INFO TaskSetManager: Finished task 3.0 in stage 11.0 (TID 34) in 9 ms on localhost (3/4)
16/04/16 15:47:45 INFO Executor: Finished task 2.0 in stage 11.0 (TID 33). 607 bytes result sent to driver
16/04/16 15:47:45 INFO TaskSetManager: Finished task 2.0 in stage 11.0 (TID 33) in 12 ms on localhost (4/4)
16/04/16 15:47:45 INFO TaskSchedulerImpl: Removed TaskSet 11.0, whose tasks have all completed, from pool
16/04/16 15:47:45 INFO DAGScheduler: ResultStage 11 (collect at <console>:20) finished in 0.013 s
16/04/16 15:47:45 INFO DAGScheduler: Job 11 finished: collect at <console>:20, took 0.022441 s
1,4,9,16
```

Example 6: union

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



```
scala> val seta = sc.parallelize(1 to 10)
16/04/16 15:50:07 INFO BlockManagerInfo: Removed broadcast_12_piece0 on localhost:58883 in memory (size: 1212.0 B, free: 265.4 MB)
16/04/16 15:50:07 INFO BlockManagerInfo: Removed broadcast_11_piece0 on localhost:58883 in memory (size: 1209.0 B, free: 265.4 MB)
seta: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[18] at parallelize at <console>:15

scala> val setb = sc.parallelize(5 to 15)
setb: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[19] at parallelize at <console>:15
```



```
scala> (seta union setb).collect
16/04/16 15:50:40 INFO SparkContext: Starting job: collect at <console>:20
16/04/16 15:50:40 INFO DAGScheduler: Got job 12 (collect at <console>:20) with 8 output partitions (allowLocal=false)

16/04/16 15:50:40 INFO TaskSetManager: Finished task 6.0 in stage 12.0 (TID 41) in 17 ms on localhost (6/8)
16/04/16 15:50:40 INFO TaskSetManager: Finished task 7.0 in stage 12.0 (TID 42) in 10 ms on localhost (7/8)
16/04/16 15:50:40 INFO TaskSetManager: Finished task 5.0 in stage 12.0 (TID 40) in 27 ms on localhost (8/8)
16/04/16 15:50:40 INFO TaskSchedulerImpl: Removed TaskSet 12.0, whose tasks have all completed, from pool
16/04/16 15:50:40 INFO DAGScheduler: ResultStage 12 (collect at <console>:20) finished in 0.047 s
16/04/16 15:50:40 INFO DAGScheduler: Job 12 finished: collect at <console>:20, took 0.071821 s
res12: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)
```

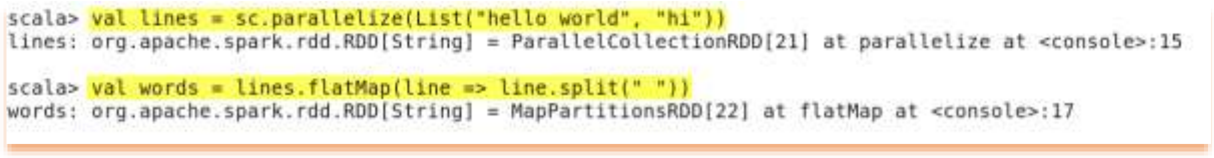
Example 7: flatMap() using scala shell

Splitting lines into multiple words

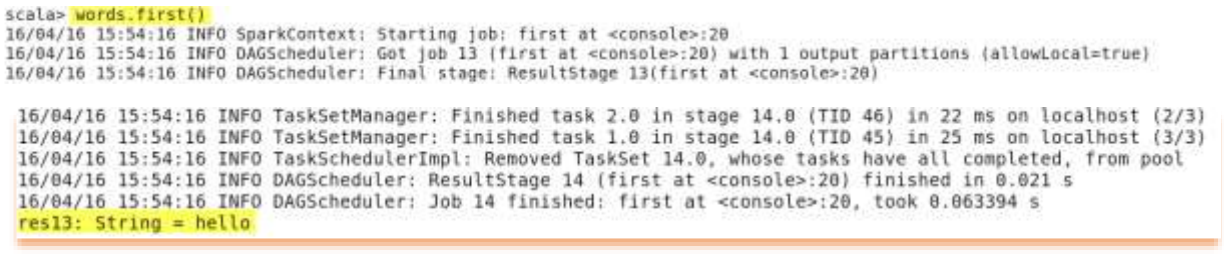
val lines = sc.parallelize(List("hello world", "hi"))

val words = lines.flatMap(line => line.split(" "))

words.first() // returns "hello"



```
scala> val lines = sc.parallelize(List("hello world", "hi"))
lines: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[21] at parallelize at <console>:15

scala> val words = lines.flatMap(line => line.split(" "))
words: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[22] at flatMap at <console>:17
```



```
scala> words.first()
16/04/16 15:54:16 INFO SparkContext: Starting job: first at <console>:20
16/04/16 15:54:16 INFO DAGScheduler: Got job 13 (first at <console>:20) with 1 output partitions (allowLocal=true)
16/04/16 15:54:16 INFO DAGScheduler: Final stage: ResultStage 13(first at <console>:20)

16/04/16 15:54:16 INFO TaskSetManager: Finished task 2.0 in stage 14.0 (TID 46) in 22 ms on localhost (2/3)
16/04/16 15:54:16 INFO TaskSetManager: Finished task 1.0 in stage 14.0 (TID 45) in 25 ms on localhost (3/3)
16/04/16 15:54:16 INFO TaskSchedulerImpl: Removed TaskSet 14.0, whose tasks have all completed, from pool
16/04/16 15:54:16 INFO DAGScheduler: ResultStage 14 (first at <console>:20) finished in 0.021 s
16/04/16 15:54:16 INFO DAGScheduler: Job 14 finished: first at <console>:20, took 0.063394 s
res13: String = hello
```

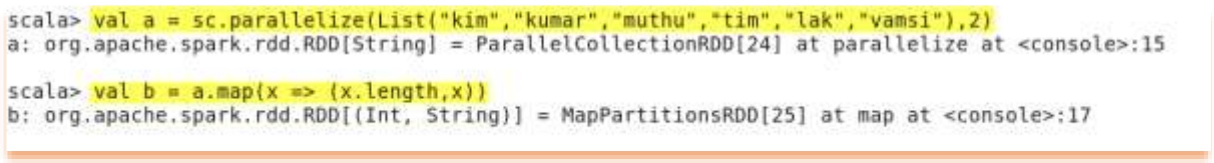
3. Paired RDD

- Spark provides special operations on RDDs containing key/value pairs. These RDDs are called pair RDD.

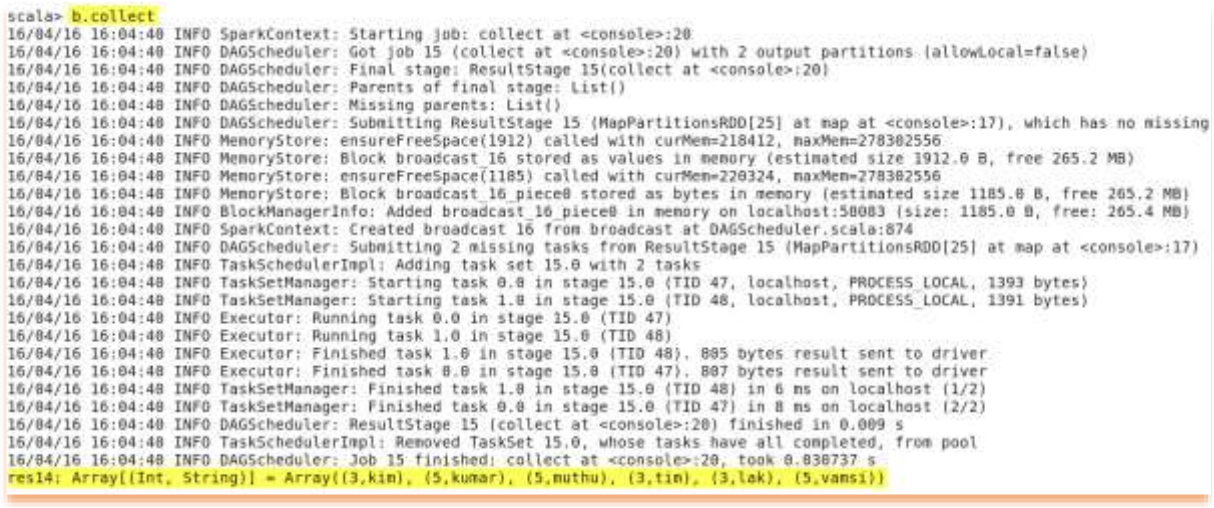
- Pair RDD allows you to act on each key in parallel or regroup data across the network.

Example 8: foldByKey

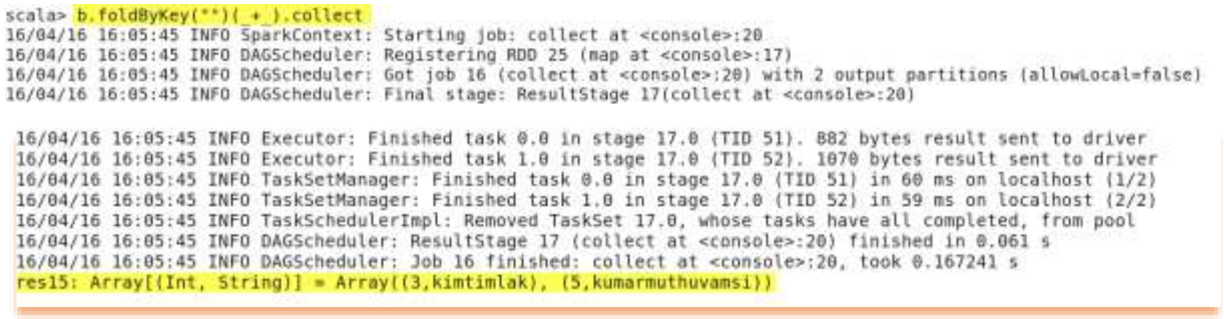
```
val a = sc.parallelize(List("kim","kumar","muthu","tim","lak","vamsi"),2)
val b = a.map(x => (x.length,x))

b.collect
b.foldByKey("")(_+_).collect
```



```
scala> val a = sc.parallelize(List("kim","kumar","muthu","tim","lak","vamsi"),2)
a: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[24] at parallelize at <console>:15

scala> val b = a.map(x => (x.length,x))
b: org.apache.spark.rdd.RDD[(Int, String)] = MapPartitionsRDD[25] at map at <console>:17
```



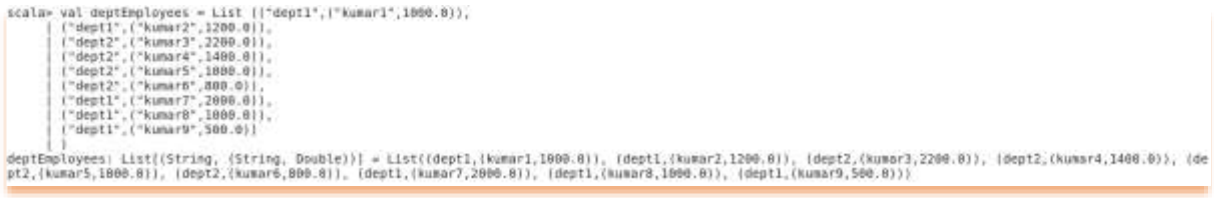
```
scala> b.collect
16/04/16 16:04:40 INFO SparkContext: Starting job: collect at <console>:20
16/04/16 16:04:40 INFO DAGScheduler: Got job 15 (collect at <console>:20) with 2 output partitions (allowLocal=false)
16/04/16 16:04:40 INFO DAGScheduler: Final stage: ResultStage 15(collect at <console>:20)
16/04/16 16:04:48 INFO DAGScheduler: Parents of final stage: List()
16/04/16 16:04:48 INFO DAGScheduler: Missing parents: List()
16/04/16 16:04:40 INFO DAGScheduler: Submitting ResultStage 15 (MapPartitionsRDD[25] at map at <console>:17), which has no missing
16/04/16 16:04:40 INFO MemoryStore: ensureFreeSpace(1912) called with curMem=218412, maxMem=278382556
16/04/16 16:04:48 INFO MemoryStore: Block broadcast_16 stored as values in memory (estimated size 1912.0 B, free 265.2 MB)
16/04/16 16:04:48 INFO MemoryStore: ensureFreeSpace(1185) called with curMem=220324, maxMem=278382556
16/04/16 16:04:48 INFO MemoryStore: Block broadcast_16_piece0 stored as bytes in memory (estimated size 1185.0 B, free 265.2 MB)
16/04/16 16:04:48 INFO BlockManagerInfo: Added broadcast_16_piece0 in memory on localhost:58083 (size: 1185.0 B, free: 265.4 MB)
16/04/16 16:04:40 INFO SparkContext: Created broadcast 16 from broadcast at DAGScheduler.scala:874
16/04/16 16:04:40 INFO DAGScheduler: Submitting 2 missing tasks from ResultStage 15 (MapPartitionsRDD[25] at map at <console>:17)
16/04/16 16:04:48 INFO TaskSchedulerImpl: Adding task set 15.0 with 2 tasks
16/04/16 16:04:48 INFO TaskSetManager: Starting task 0.0 in stage 15.0 (TID 47, localhost, PROCESS_LOCAL, 1393 bytes)
16/04/16 16:04:48 INFO TaskSetManager: Starting task 1.0 in stage 15.0 (TID 48, localhost, PROCESS_LOCAL, 1391 bytes)
16/04/16 16:04:48 INFO Executor: Running task 0.0 in stage 15.0 (TID 47)
16/04/16 16:04:40 INFO Executor: Running task 1.0 in stage 15.0 (TID 48)
16/04/16 16:04:40 INFO Executor: Finished task 1.0 in stage 15.0 (TID 48). 805 bytes result sent to driver
16/04/16 16:04:48 INFO Executor: Finished task 0.0 in stage 15.0 (TID 47). 807 bytes result sent to driver
16/04/16 16:04:48 INFO TaskSetManager: Finished task 1.0 in stage 15.0 (TID 48) in 6 ms on localhost (1/2)
16/04/16 16:04:48 INFO TaskSetManager: Finished task 0.0 in stage 15.0 (TID 47) in 8 ms on localhost (2/2)
16/04/16 16:04:40 INFO DAGScheduler: ResultStage 15 (collect at <console>:20) finished in 0.009 s
16/04/16 16:04:48 INFO TaskSchedulerImpl: Removed TaskSet 15.0, whose tasks have all completed, from pool
16/04/16 16:04:48 INFO DAGScheduler: Job 15 finished: collect at <console>:20, took 0.038737 s
res14: Array[(Int, String)] = Array((3,kim), (5,kumar), (5,muthu), (3,tim), (3,lak), (5,vamsi))
```



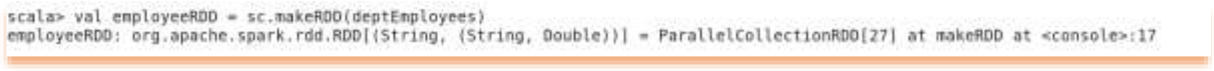
```
scala> b.foldByKey("")(_+_).collect
16/04/16 16:05:45 INFO SparkContext: Starting job: collect at <console>:20
16/04/16 16:05:45 INFO DAGScheduler: Registering RDD 25 (map at <console>:17)
16/04/16 16:05:45 INFO DAGScheduler: Got job 16 (collect at <console>:20) with 2 output partitions (allowLocal=false)
16/04/16 16:05:45 INFO DAGScheduler: Final stage: ResultStage 17(collect at <console>:20)

16/04/16 16:05:45 INFO Executor: Finished task 0.0 in stage 17.0 (TID 51). 882 bytes result sent to driver
16/04/16 16:05:45 INFO Executor: Finished task 1.0 in stage 17.0 (TID 52). 1070 bytes result sent to driver
16/04/16 16:05:45 INFO TaskSetManager: Finished task 0.0 in stage 17.0 (TID 51) in 60 ms on localhost (1/2)
16/04/16 16:05:45 INFO TaskSetManager: Finished task 1.0 in stage 17.0 (TID 52) in 59 ms on localhost (2/2)
16/04/16 16:05:45 INFO TaskSchedulerImpl: Removed TaskSet 17.0, whose tasks have all completed, from pool
16/04/16 16:05:45 INFO DAGScheduler: ResultStage 17 (collect at <console>:20) finished in 0.061 s
16/04/16 16:05:45 INFO DAGScheduler: Job 16 finished: collect at <console>:20, took 0.167241 s
res15: Array[(Int, String)] = Array((3,kimtimlak), (5,kumarmuthuvamsi))
```

Example 9: foldByKey

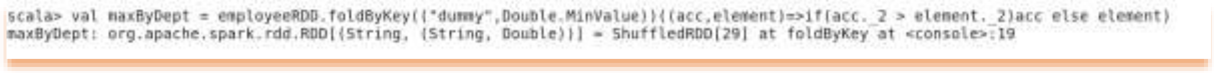
```
val deptEmployees =
  List
  (
   ("dept1",("kumar1",1000.0)),
   ("dept1",("kumar2",1200.0)),
   ("dept2",("kumar3",2200.0)),
   ("dept2",("kumar4",1400.0)),
   ("dept2",("kumar5",1000.0)),
   ("dept2",("kumar6",800.0)),
   ("dept1",("kumar7",2000.0)),
   ("dept1",("kumar8",1000.0)),
   ("dept1",("kumar9",500.0))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



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)
```



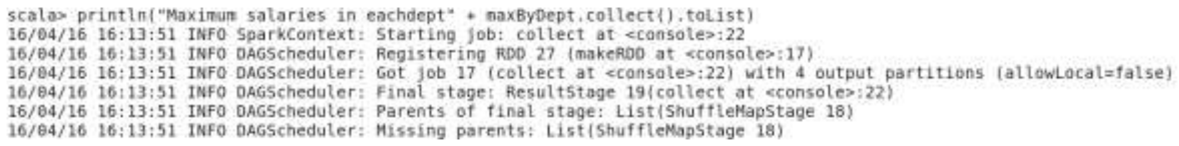
```
scala> val deptEmployees = List (("dept1",("kumar1",1000.0)),
     | ("dept1",("kumar2",1200.0)),
     | ("dept2",("kumar3",2200.0)),
     | ("dept2",("kumar4",1400.0)),
     | ("dept2",("kumar5",1000.0)),
     | ("dept2",("kumar6",800.0)),
     | ("dept1",("kumar7",2000.0)),
     | ("dept1",("kumar8",1000.0)),
     | ("dept1",("kumar9",500.0))
     | )
deptEmployees: List[(String, (String, Double))] = List((dept1,(kumar1,1000.0)), (dept1,(kumar2,1200.0)), (dept2,(kumar3,2200.0)), (dept2,(kumar4,1400.0)), (de
pt2,(kumar5,1000.0)), (dept2,(kumar6,800.0)), (dept1,(kumar7,2000.0)), (dept1,(kumar8,1000.0)), (dept1,(kumar9,500.0)))
```



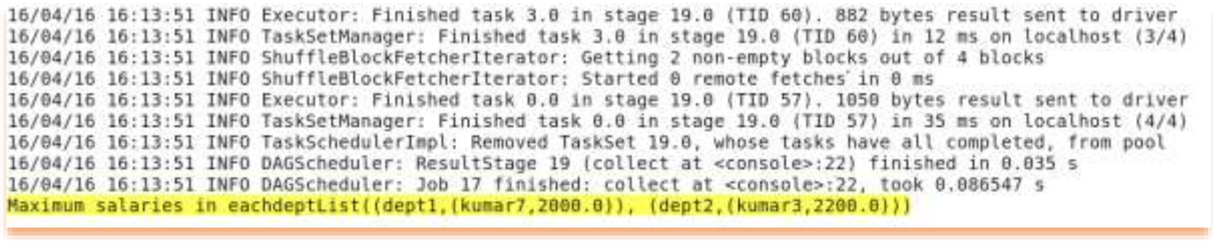
```
scala> val employeeRDD = sc.makeRDD(deptEmployees)
employeeRDD: org.apache.spark.rdd.RDD[(String, (String, Double))] = ParallelCollectionRDD[27] at makeRDD at <console>:17
```



```
scala> val maxByDept = employeeRDD.foldByKey(("dummy",Double.MinValue))((acc,element)=>if(acc._2 > element._2)acc else element)
maxByDept: org.apache.spark.rdd.RDD[(String, (String, Double))] = ShuffledRDD[29] at foldByKey at <console>:19
```



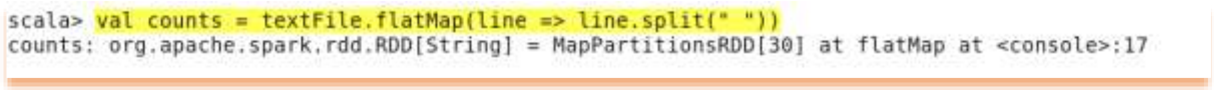
```
scala> println("Maximum salaries in eachdept" + maxByDept.collect().toList)
16/04/16 16:13:51 INFO SparkContext: Starting job: collect at <console>:22
16/04/16 16:13:51 INFO DAGScheduler: Registering RDD 27 (makeRDD at <console>:17)
16/04/16 16:13:51 INFO DAGScheduler: Got job 17 (collect at <console>:22) with 4 output partitions (allowLocal=false)
16/04/16 16:13:51 INFO DAGScheduler: Final stage: ResultStage 19(collect at <console>:22)
16/04/16 16:13:51 INFO DAGScheduler: Parents of final stage: List(ShuffleMapStage 18)
16/04/16 16:13:51 INFO DAGScheduler: Missing parents: List(ShuffleMapStage 18)
```



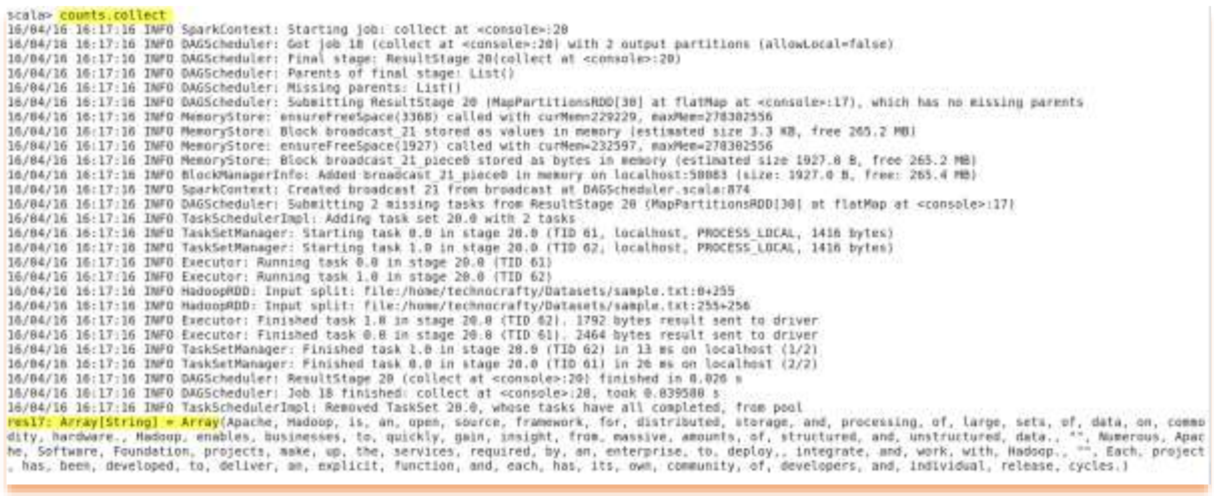
```
16/04/16 16:13:51 INFO Executor: Finished task 3.0 in stage 19.0 (TID 60). 882 bytes result sent to driver
16/04/16 16:13:51 INFO TaskSetManager: Finished task 3.0 in stage 19.0 (TID 60) in 12 ms on localhost (3/4)
16/04/16 16:13:51 INFO ShuffleBlockFetcherIterator: Getting 2 non-empty blocks out of 4 blocks
16/04/16 16:13:51 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
16/04/16 16:13:51 INFO Executor: Finished task 0.0 in stage 19.0 (TID 57). 1050 bytes result sent to driver
16/04/16 16:13:51 INFO TaskSetManager: Finished task 0.0 in stage 19.0 (TID 57) in 35 ms on localhost (4/4)
16/04/16 16:13:51 INFO TaskSchedulerImpl: Removed TaskSet 19.0, whose tasks have all completed, from pool
16/04/16 16:13:51 INFO DAGScheduler: ResultStage 19 (collect at <console>:22) finished in 0.035 s
16/04/16 16:13:51 INFO DAGScheduler: Job 17 finished: collect at <console>:22, took 0.086547 s
Maximum salaries in eachdeptList((dept1,(kumar7,2000.0)), (dept2,(kumar3,2200.0)))
```

Example 10: reduceByKey

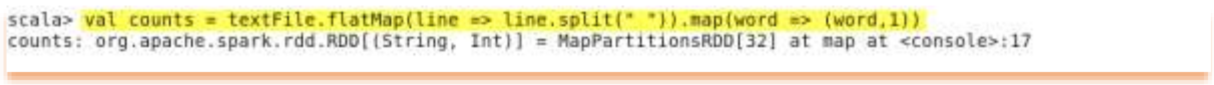
```
val textFile = sc.textFile("file:/home/technocrafty/Datasets/sample.txt ")

val counts = textFile.flatMap(line => line.split(" "))
counts.collect
val counts = textFile.flatMap(line => line.split(" ")).map(word => (word,1))
counts.collect
val counts = textFile.flatMap(line => line.split(" ")).map(word     =>
(word,1)).reduceByKey(_+_)
counts.collect.foreach(println)
```



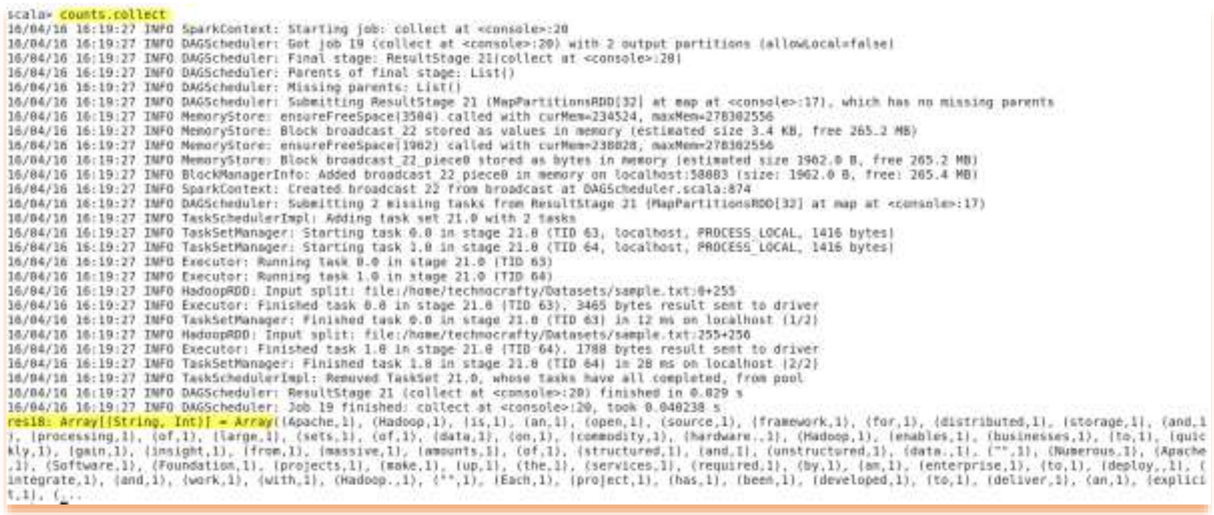
```
scala> val counts = textFile.flatMap(line => line.split(" "))
counts: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[30] at flatMap at <console>:17
```



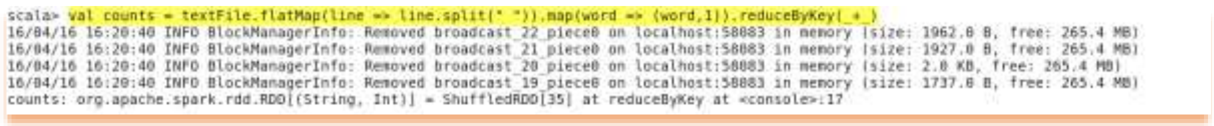
```
scala> counts.collect
16/04/16 16:17:16 INFO SparkContext: Starting job: collect at <console>:20
16/04/16 16:17:16 INFO DAGScheduler: Got job 18 (collect at <console>:20) with 2 output partitions (allowLocal=false)
16/04/16 16:17:16 INFO DAGScheduler: Final stage: ResultStage 20(collect at <console>:20)
16/04/16 16:17:16 INFO DAGScheduler: Parents of final stage: List()
16/04/16 16:17:16 INFO DAGScheduler: Missing parents: List()
16/04/16 16:17:16 INFO DAGScheduler: Submitting ResultStage 20 (MapPartitionsRDD[30] at flatMap at <console>:17), which has no missing parents
16/04/16 16:17:16 INFO MemoryStore: ensureFreeSpace(3368) called with curMem=229229, maxMem=278302556
16/04/16 16:17:16 INFO MemoryStore: Block broadcast_21 stored as values in memory (estimated size 3.3 KB, free 265.2 MB)
16/04/16 16:17:16 INFO MemoryStore: ensureFreeSpace(1927) called with curMem=232597, maxMem=278302556
16/04/16 16:17:16 INFO MemoryStore: Block broadcast_21_piece0 stored as bytes in memory (estimated size 1927.0 B, free 265.2 MB)
16/04/16 16:17:16 INFO BlockManagerInfo: Added broadcast_21_piece0 in memory on localhost:58083 (size: 1927.0 B, free: 265.4 MB)
16/04/16 16:17:16 INFO SparkContext: Created broadcast 21 from broadcast at DAGScheduler.scala:874
16/04/16 16:17:16 INFO DAGScheduler: Submitting 2 missing tasks from ResultStage 20 (MapPartitionsRDD[30] at flatMap at <console>:17)
16/04/16 16:17:16 INFO TaskSchedulerImpl: Adding task set 20.0 with 2 tasks
16/04/16 16:17:16 INFO TaskSetManager: Starting task 0.0 in stage 20.0 (TID 61, localhost, PROCESS_LOCAL, 1416 bytes)
16/04/16 16:17:16 INFO TaskSetManager: Starting task 1.0 in stage 20.0 (TID 62, localhost, PROCESS_LOCAL, 1416 bytes)
16/04/16 16:17:16 INFO Executor: Running task 0.0 in stage 20.0 (TID 61)
16/04/16 16:17:16 INFO Executor: Running task 1.0 in stage 20.0 (TID 62)
16/04/16 16:17:16 INFO HadoopRDD: Input split: file:/home/technocrafty/Datasets/sample.txt:0+255
16/04/16 16:17:16 INFO HadoopRDD: Input split: file:/home/technocrafty/Datasets/sample.txt:255+256
16/04/16 16:17:16 INFO Executor: Finished task 1.0 in stage 20.0 (TID 62). 1792 bytes result sent to driver
16/04/16 16:17:16 INFO Executor: Finished task 0.0 in stage 20.0 (TID 61). 2464 bytes result sent to driver
16/04/16 16:17:16 INFO TaskSetManager: Finished task 1.0 in stage 20.0 (TID 62) in 13 ms on localhost (1/2)
16/04/16 16:17:16 INFO TaskSetManager: Finished task 0.0 in stage 20.0 (TID 61) in 26 ms on localhost (2/2)
16/04/16 16:17:16 INFO DAGScheduler: ResultStage 20 (collect at <console>:20) finished in 0.026 s
16/04/16 16:17:16 INFO DAGScheduler: Job 18 finished: collect at <console>:20, took 0.039580 s
16/04/16 16:17:16 INFO TaskSchedulerImpl: Removed TaskSet 20.0, whose tasks have all completed, from pool
res17: Array[String] = Array(Apache, Hadoop, is, an, open, source, framework, for, distributed, storage, and, processing, of, large, sets, of, data, on, commo
dity, hardware., Hadoop, enables, businesses, to, quickly, gain, insight, from, massive, amounts, of, structured, and, unstructured, data., "", Numerous, Apac
he, Software, Foundation, projects, make, up, the, services, required, by, an, enterprise, to, deploy,, integrate, and, work, with, Hadoop., "", Each, project
, has, been, developed, to, deliver, an, explicit, function, and, each, has, its, own, community, of, developers, and, individual, release, cycles.)
```



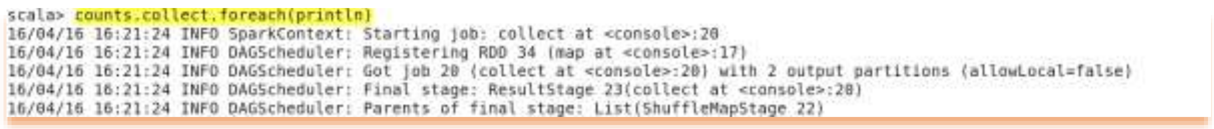
```
scala> val counts = textFile.flatMap(line => line.split(" ")).map(word => (word,1))
counts: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[32] at map at <console>:17
```



```
scala> counts.collect
16/04/16 16:19:27 INFO SparkContext: Starting job: collect at <console>:20
16/04/16 16:19:27 INFO DAGScheduler: Got job 19 (collect at <console>:20) with 2 output partitions (allowLocal=false)
16/04/16 16:19:27 INFO DAGScheduler: Final stage: ResultStage 21(collect at <console>:20)
16/04/16 16:19:27 INFO DAGScheduler: Parents of final stage: List()
16/04/16 16:19:27 INFO DAGScheduler: Missing parents: List()
16/04/16 16:19:27 INFO DAGScheduler: Submitting ResultStage 21 (MapPartitionsRDD[32] at map at <console>:17), which has no missing parents
16/04/16 16:19:27 INFO MemoryStore: ensureFreeSpace(3504) called with curMem=234524, maxMem=278302556
16/04/16 16:19:27 INFO MemoryStore: Block broadcast 22 stored as values in memory (estimated size 3.4 KB, free 265.2 MB)
16/04/16 16:19:27 INFO MemoryStore: ensureFreeSpace(1962) called with curMem=238028, maxMem=278302556
16/04/16 16:19:27 INFO MemoryStore: Block broadcast_22_piece0 stored as bytes in memory (estimated size 1962.0 B, free 265.2 MB)
16/04/16 16:19:27 INFO BlockManagerInfo: Added broadcast 22 piece0 in memory on localhost:58883 (size: 1962.0 B, free: 265.4 MB)
16/04/16 16:19:27 INFO SparkContext: Created broadcast 22 from broadcast at DAGScheduler.scala:874
16/04/16 16:19:27 INFO DAGScheduler: Submitting 2 missing tasks from ResultStage 21 (MapPartitionsRDD[32] at map at <console>:17)
16/04/16 16:19:27 INFO TaskSchedulerImpl: Adding task set 21.0 with 2 tasks
16/04/16 16:19:27 INFO TaskSetManager: Starting task 0.0 in stage 21.0 (TID 63, localhost, PROCESS_LOCAL, 1416 bytes)
16/04/16 16:19:27 INFO TaskSetManager: Starting task 1.0 in stage 21.0 (TID 64, localhost, PROCESS_LOCAL, 1416 bytes)
16/04/16 16:19:27 INFO Executor: Running task 0.0 in stage 21.0 (TID 63)
16/04/16 16:19:27 INFO Executor: Running task 1.0 in stage 21.0 (TID 64)
16/04/16 16:19:27 INFO HadoopRDD: Input split: file:/home/technocrafty/Datasets/sample.txt:0+255
16/04/16 16:19:27 INFO Executor: Finished task 0.0 in stage 21.0 (TID 63). 3465 bytes result sent to driver
16/04/16 16:19:27 INFO TaskSetManager: Finished task 0.0 in stage 21.0 (TID 63) in 12 ms on localhost (1/2)
16/04/16 16:19:27 INFO HadoopRDD: Input split: file:/home/technocrafty/Datasets/sample.txt:255+256
16/04/16 16:19:27 INFO Executor: Finished task 1.0 in stage 21.0 (TID 64). 1788 bytes result sent to driver
16/04/16 16:19:27 INFO TaskSetManager: Finished task 1.0 in stage 21.0 (TID 64) in 28 ms on localhost (2/2)
16/04/16 16:19:27 INFO TaskSchedulerImpl: Removed TaskSet 21.0, whose tasks have all completed, from pool
16/04/16 16:19:27 INFO DAGScheduler: ResultStage 21 (collect at <console>:20) finished in 0.029 s
16/04/16 16:19:27 INFO DAGScheduler: Job 19 finished: collect at <console>:20, took 0.040238 s
res18: Array[(String, Int)] = Array((Apache,1), (Hadoop,1), (is,1), (an,1), (open,1), (source,1), (framework,1), (for,1), (distributed,1), (storage,1), (and,1
), (processing,1), (of,1), (large,1), (sets,1), (of,1), (data,1), (on,1), (commodity,1), (hardware.,1), (Hadoop,1), (enables,1), (businesses,1), (to,1), (quic
kly,1), (gain,1), (insight,1), (from,1), (massive,1), (amounts,1), (of,1), (structured,1), (and,1), (unstructured,1), (data.,1), ("",1), (Numerous,1), (Apache
,1), (Software,1), (Foundation,1), (projects,1), (make,1), (up,1), (the,1), (services,1), (required,1), (by,1), (an,1), (enterprise,1), (to,1), (deploy,,1), (
integrate,1), (and,1), (work,1), (with,1), (Hadoop.,1), ("",1), (Each,1), (project,1), (has,1), (been,1), (developed,1), (to,1), (deliver,1), (an,1), (explici
t,1), (...
```

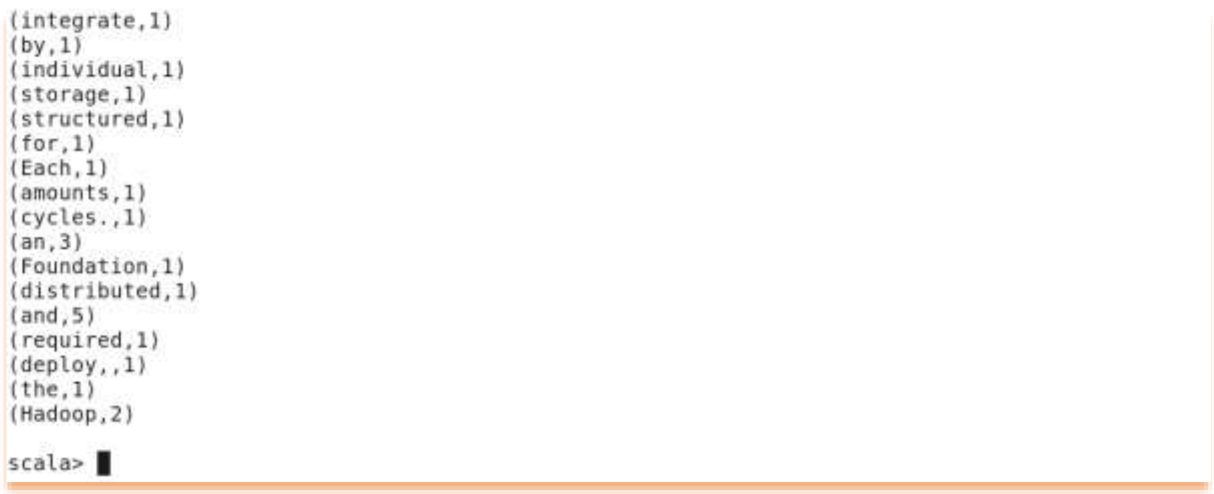


```
scala> val counts = textFile.flatMap(line => line.split(" ")).map(word => (word,1)).reduceByKey(_+_)
16/04/16 16:20:40 INFO BlockManagerInfo: Removed broadcast_22_piece0 on localhost:58883 in memory (size: 1962.0 B, free: 265.4 MB)
16/04/16 16:20:40 INFO BlockManagerInfo: Removed broadcast_21_piece0 on localhost:58883 in memory (size: 1927.0 B, free: 265.4 MB)
16/04/16 16:20:40 INFO BlockManagerInfo: Removed broadcast_20_piece0 on localhost:58883 in memory (size: 2.0 KB, free: 265.4 MB)
16/04/16 16:20:40 INFO BlockManagerInfo: Removed broadcast_19_piece0 on localhost:58883 in memory (size: 1737.0 B, free: 265.4 MB)
counts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[35] at reduceByKey at <console>:17
```



```
scala> counts.collect.foreach(println)
16/04/16 16:21:24 INFO SparkContext: Starting job: collect at <console>:20
16/04/16 16:21:24 INFO DAGScheduler: Registering RDD 34 (map at <console>:17)
16/04/16 16:21:24 INFO DAGScheduler: Got job 20 (collect at <console>:20) with 2 output partitions (allowLocal=false)
16/04/16 16:21:24 INFO DAGScheduler: Final stage: ResultStage 23(collect at <console>:20)
16/04/16 16:21:24 INFO DAGScheduler: Parents of final stage: List(ShuffleMapStage 22)
```

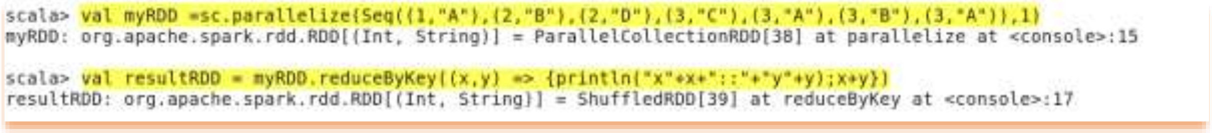
Final print output logs are too big to display here, but the content will look as below



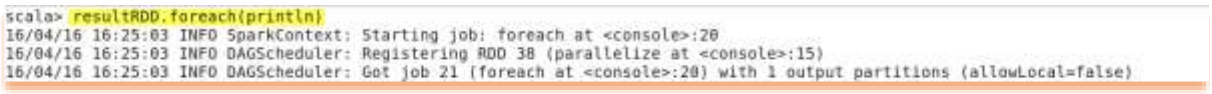
```
(integrate,1)
(by,1)
(individual,1)
(storage,1)
(structured,1)
(for,1)
(Each,1)
(amounts,1)
(cycles.,1)
(an,3)
(Foundation,1)
(distributed,1)
(and,5)
(required,1)
(deploy,,1)
(the,1)
(Hadoop,2)

scala> 
```

Example 11: reduceByKey

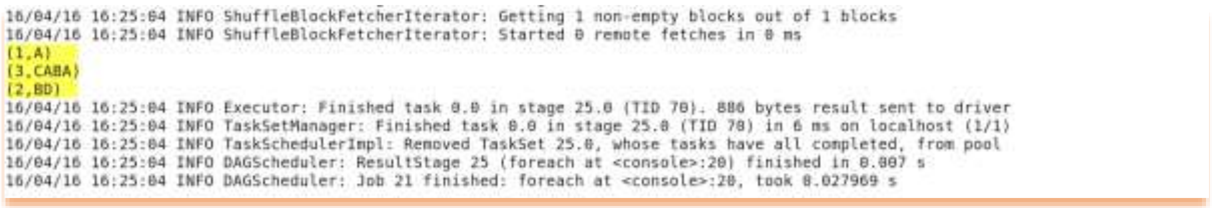
```
val myRDD = sc.parallelize(Seq((1,"A"),(2,"B"),(2,"D"),(3,"C"),(3,"A"),(3,"B"),(3,"A")),1)
val resultRDD = myRDD.reduceByKey((x, y) => {println("x"+x+"::"+"y"+y);x+y})
resultRDD.foreach(println)
```



```
scala> val myRDD =sc.parallelize(Seq((1,"A"),(2,"B"),(2,"D"),(3,"C"),(3,"A"),(3,"B"),(3,"A")),1)
myRDD: org.apache.spark.rdd.RDD[(Int, String)] = ParallelCollectionRDD[38] at parallelize at <console>:15

scala> val resultRDD = myRDD.reduceByKey((x,y) => {println("x"+x+"::"+"y"+y);x+y})
resultRDD: org.apache.spark.rdd.RDD[(Int, String)] = ShuffledRDD[39] at reduceByKey at <console>:17
```



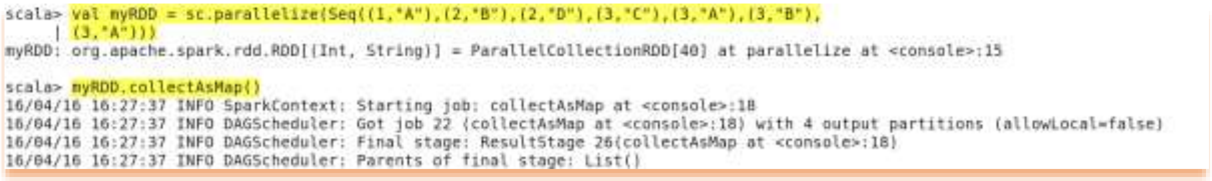
```
scala> resultRDD.foreach(println)
16/04/16 16:25:03 INFO SparkContext: Starting job: foreach at <console>:20
16/04/16 16:25:03 INFO DAGScheduler: Registering RDD 38 (parallelize at <console>:15)
16/04/16 16:25:03 INFO DAGScheduler: Got job 21 (foreach at <console>:20) with 1 output partitions (allowLocal=false)
```



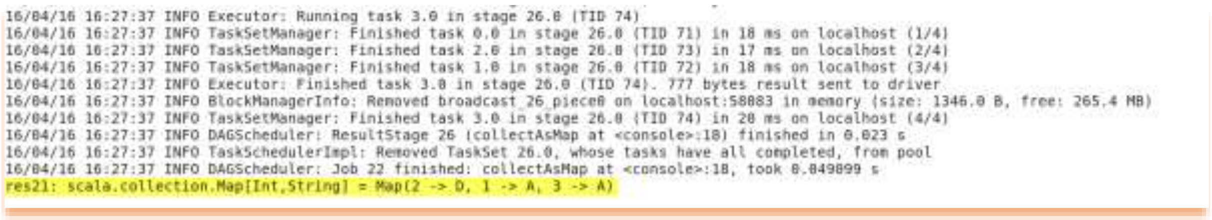
```
16/04/16 16:25:04 INFO ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of 1 blocks
16/04/16 16:25:04 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
(1,A)
(3,CABA)
(2,BD)
16/04/16 16:25:04 INFO Executor: Finished task 0.0 in stage 25.0 (TID 70). 886 bytes result sent to driver
16/04/16 16:25:04 INFO TaskSetManager: Finished task 0.0 in stage 25.0 (TID 70) in 6 ms on localhost (1/1)
16/04/16 16:25:04 INFO TaskSchedulerImpl: Removed TaskSet 25.0, whose tasks have all completed, from pool
16/04/16 16:25:04 INFO DAGScheduler: ResultStage 25 (foreach at <console>:20) finished in 0.007 s
16/04/16 16:25:04 INFO DAGScheduler: Job 21 finished: foreach at <console>:20, took 0.027969 s
```

Example 12: collectAsMap

```
val myRDD = sc.parallelize(Seq((1,"A"),(2,"B"),(2,"D"),(3,"C"),(3,"A"),(3,"B"),

(3,"A")))
myRDD.collectAsMap()
```



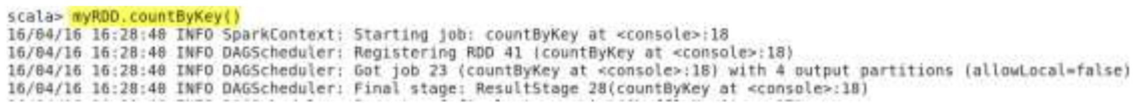
```
scala> val myRDD = sc.parallelize(Seq((1,"A"),(2,"B"),(2,"D"),(3,"C"),(3,"A"),(3,"B"),
     | (3,"A")))
myRDD: org.apache.spark.rdd.RDD[(Int, String)] = ParallelCollectionRDD[40] at parallelize at <console>:15

scala> myRDD.collectAsMap()
16/04/16 16:27:37 INFO SparkContext: Starting job: collectAsMap at <console>:18
16/04/16 16:27:37 INFO DAGScheduler: Got job 22 (collectAsMap at <console>:18) with 4 output partitions (allowLocal=false)
16/04/16 16:27:37 INFO DAGScheduler: Final stage: ResultStage 26(collectAsMap at <console>:18)
16/04/16 16:27:37 INFO DAGScheduler: Parents of final stage: List()
```



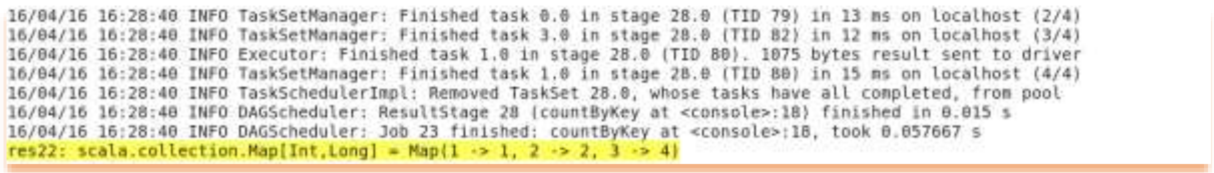
```
16/04/16 16:27:37 INFO Executor: Running task 3.0 in stage 26.0 (TID 74)
16/04/16 16:27:37 INFO TaskSetManager: Finished task 0.0 in stage 26.0 (TID 71) in 18 ms on localhost (1/4)
16/04/16 16:27:37 INFO TaskSetManager: Finished task 2.0 in stage 26.0 (TID 73) in 17 ms on localhost (2/4)
16/04/16 16:27:37 INFO TaskSetManager: Finished task 1.0 in stage 26.0 (TID 72) in 18 ms on localhost (3/4)
16/04/16 16:27:37 INFO Executor: Finished task 3.0 in stage 26.0 (TID 74). 777 bytes result sent to driver
16/04/16 16:27:37 INFO BlockManagerInfo: Removed broadcast_26_piece0 on localhost:58883 in memory (size: 1346.0 B, free: 265.4 MB)
16/04/16 16:27:37 INFO TaskSetManager: Finished task 3.0 in stage 26.0 (TID 74) in 20 ms on localhost (4/4)
16/04/16 16:27:37 INFO DAGScheduler: ResultStage 26 (collectAsMap at <console>:18) finished in 0.023 s
16/04/16 16:27:37 INFO TaskSchedulerImpl: Removed TaskSet 26.0, whose tasks have all completed, from pool
16/04/16 16:27:37 INFO DAGScheduler: Job 22 finished: collectAsMap at <console>:18, took 0.049899 s
res21: scala.collection.Map[Int,String] = Map(2 -> D, 1 -> A, 3 -> A)
```

Example 13: countByKey

```
myRDD.countByKey()
```



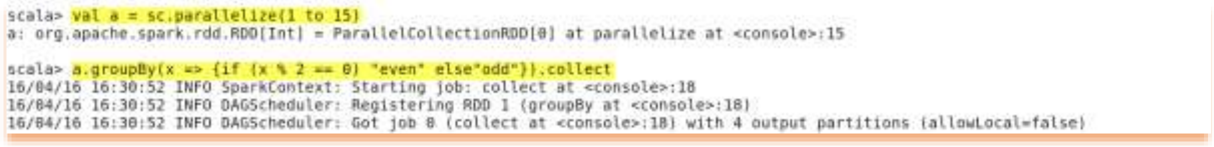
```
scala> myRDD.countByKey()
16/04/16 16:28:40 INFO SparkContext: Starting job: countByKey at <console>:18
16/04/16 16:28:40 INFO DAGScheduler: Registering RDD 41 (countByKey at <console>:18)
16/04/16 16:28:40 INFO DAGScheduler: Got job 23 (countByKey at <console>:18) with 4 output partitions (allowLocal=false)
16/04/16 16:28:40 INFO DAGScheduler: Final stage: ResultStage 28(countByKey at <console>:18)
```



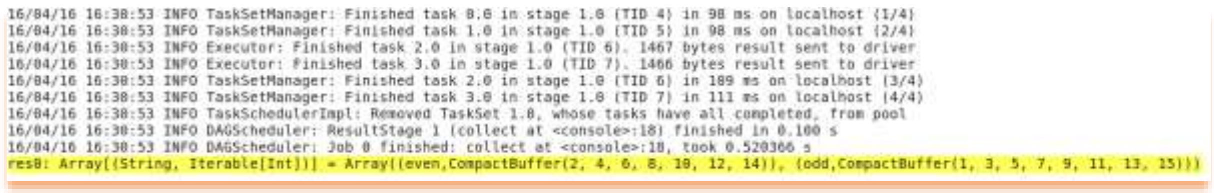
```
16/04/16 16:28:40 INFO TaskSetManager: Finished task 0.0 in stage 28.0 (TID 79) in 13 ms on localhost (2/4)
16/04/16 16:28:40 INFO TaskSetManager: Finished task 3.0 in stage 28.0 (TID 82) in 12 ms on localhost (3/4)
16/04/16 16:28:40 INFO Executor: Finished task 1.0 in stage 28.0 (TID 80). 1075 bytes result sent to driver
16/04/16 16:28:40 INFO TaskSetManager: Finished task 1.0 in stage 28.0 (TID 80) in 15 ms on localhost (4/4)
16/04/16 16:28:40 INFO TaskSchedulerImpl: Removed TaskSet 28.0, whose tasks have all completed, from pool
16/04/16 16:28:40 INFO DAGScheduler: ResultStage 28 (countByKey at <console>:18) finished in 0.015 s
16/04/16 16:28:40 INFO DAGScheduler: Job 23 finished: countByKey at <console>:18, took 0.057667 s
res22: scala.collection.Map[Int,Long] = Map(1 -> 1, 2 -> 2, 3 -> 4)
```

Example 14: groupBy

```
val a = sc.parallelize(1 to 15)
a.groupBy(x => {if (x % 2 == 0) "even" else"odd"}).collect
```



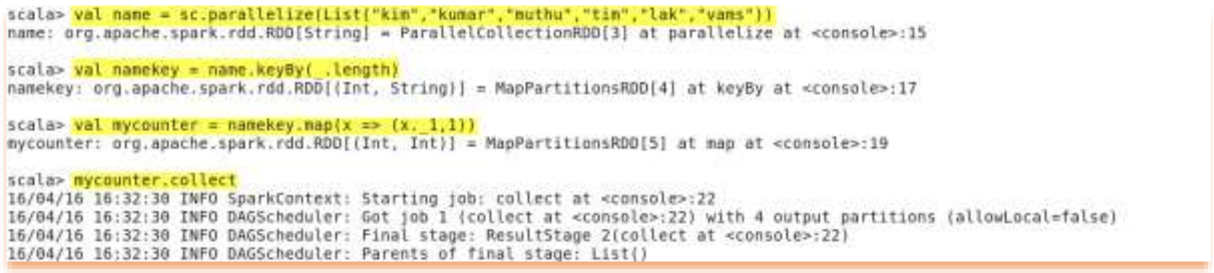
```
scala> val a = sc.parallelize(1 to 15)
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:15

scala> a.groupBy(x => {if (x % 2 == 0) "even" else"odd"}).collect
16/04/16 16:30:52 INFO SparkContext: Starting job: collect at <console>:18
16/04/16 16:30:52 INFO DAGScheduler: Registering RDD 1 (groupBy at <console>:18)
16/04/16 16:30:52 INFO DAGScheduler: Got job 0 (collect at <console>:18) with 4 output partitions (allowLocal=false)
```



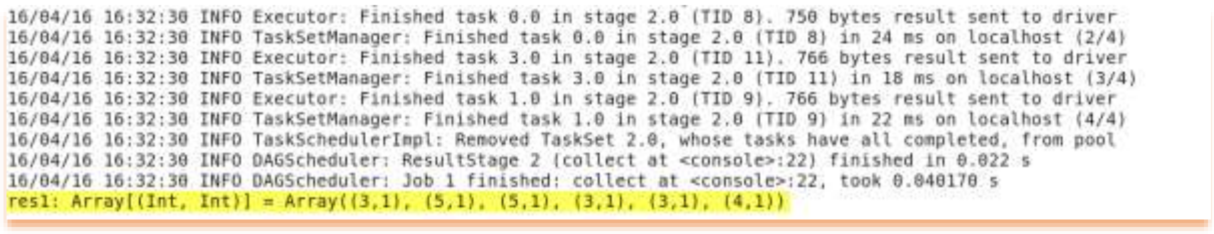
```
16/04/16 16:30:53 INFO TaskSetManager: Finished task 0.0 in stage 1.0 (TID 4) in 98 ms on localhost (1/4)
16/04/16 16:30:53 INFO TaskSetManager: Finished task 1.0 in stage 1.0 (TID 5) in 98 ms on localhost (2/4)
16/04/16 16:30:53 INFO Executor: Finished task 2.0 in stage 1.0 (TID 6). 1467 bytes result sent to driver
16/04/16 16:30:53 INFO Executor: Finished task 3.0 in stage 1.0 (TID 7). 1466 bytes result sent to driver
16/04/16 16:30:53 INFO TaskSetManager: Finished task 2.0 in stage 1.0 (TID 6) in 109 ms on localhost (3/4)
16/04/16 16:30:53 INFO TaskSetManager: Finished task 3.0 in stage 1.0 (TID 7) in 111 ms on localhost (4/4)
16/04/16 16:30:53 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
16/04/16 16:30:53 INFO DAGScheduler: ResultStage 1 (collect at <console>:18) finished in 0.100 s
16/04/16 16:30:53 INFO DAGScheduler: Job 0 finished: collect at <console>:18, took 0.520366 s
res0: Array[(String, Iterable[Int])] = Array((even,CompactBuffer(2, 4, 6, 8, 10, 12, 14)), (odd,CompactBuffer(1, 3, 5, 7, 9, 11, 13, 15)))
```

Example 15: groupByKey

```
val name = sc.parallelize(List("kim","kumar","muthu","tim","lak","vams"))
val namekey = name.keyBy(_.length)
val mycounter = namekey.map(x => (x._1,1))
mycounter.collect
```



```
scala> val name = sc.parallelize(List("kim","kumar","muthu","tim","lak","vams"))
name: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[3] at parallelize at <console>:15

scala> val namekey = name.keyBy(_.length)
namekey: org.apache.spark.rdd.RDD[(Int, String)] = MapPartitionsRDD[4] at keyBy at <console>:17

scala> val mycounter = namekey.map(x => (x._1,1))
mycounter: org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[5] at map at <console>:19

scala> mycounter.collect
16/04/16 16:32:30 INFO SparkContext: Starting job: collect at <console>:22
16/04/16 16:32:30 INFO DAGScheduler: Got job 1 (collect at <console>:22) with 4 output partitions (allowLocal=false)
16/04/16 16:32:30 INFO DAGScheduler: Final stage: ResultStage 2(collect at <console>:22)
16/04/16 16:32:30 INFO DAGScheduler: Parents of final stage: List()
```



```
16/04/16 16:32:30 INFO Executor: Finished task 0.0 in stage 2.0 (TID 8). 750 bytes result sent to driver
16/04/16 16:32:30 INFO TaskSetManager: Finished task 0.0 in stage 2.0 (TID 8) in 24 ms on localhost (2/4)
16/04/16 16:32:30 INFO Executor: Finished task 3.0 in stage 2.0 (TID 11). 766 bytes result sent to driver
16/04/16 16:32:30 INFO TaskSetManager: Finished task 3.0 in stage 2.0 (TID 11) in 18 ms on localhost (3/4)
16/04/16 16:32:30 INFO Executor: Finished task 1.0 in stage 2.0 (TID 9). 766 bytes result sent to driver
16/04/16 16:32:30 INFO TaskSetManager: Finished task 1.0 in stage 2.0 (TID 9) in 22 ms on localhost (4/4)
16/04/16 16:32:30 INFO TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all completed, from pool
16/04/16 16:32:30 INFO DAGScheduler: ResultStage 2 (collect at <console>:22) finished in 0.022 s
16/04/16 16:32:30 INFO DAGScheduler: Job 1 finished: collect at <console>:22, took 0.040170 s
res1: Array[(Int, Int)] = Array((3,1), (5,1), (5,1), (3,1), (3,1), (4,1))
```

Example 16: Join

val name = sc.parallelize(List("kim","kumar","muthu","tim","lak","vams"))
val namekey = name.keyBy(_.length)

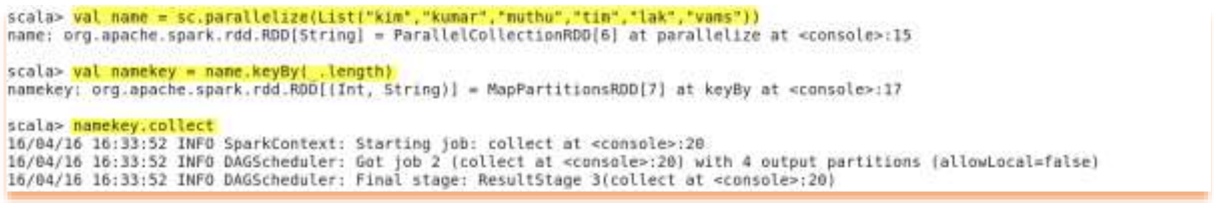
namekey.collect

val sub = sc.parallelize(List("English","Maths","Tamil","Science"))

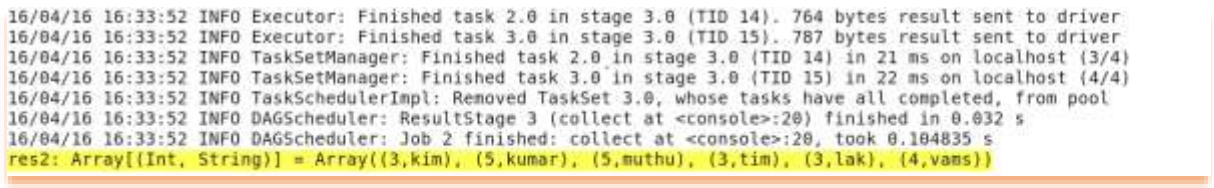
val subkey = sub.keyBy(_.length)

subkey.collect

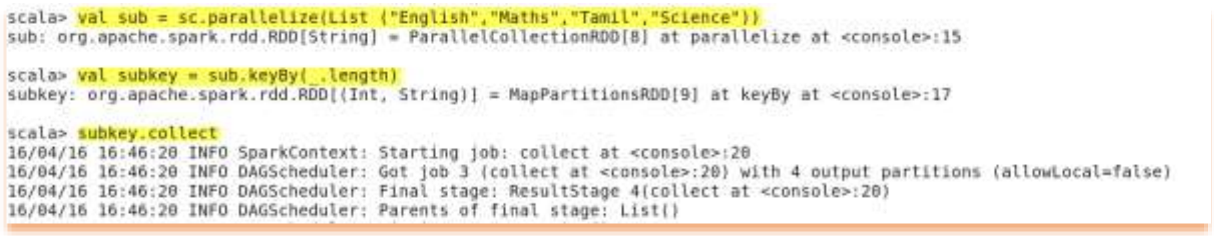
namekey.join(subkey).collect



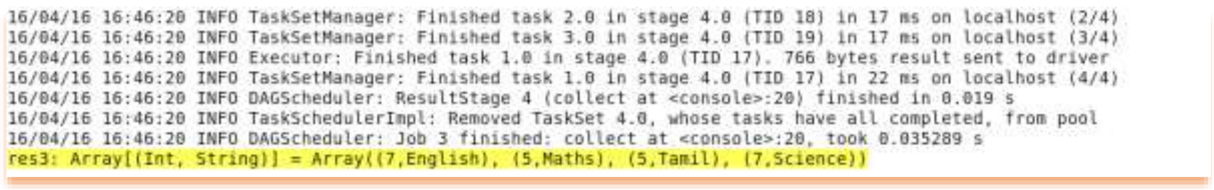
```
scala> val name = sc.parallelize(List("kim","kumar","muthu","tim","lak","vams"))
name: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[6] at parallelize at <console>:15

scala> val namekey = name.keyBy(_.length)
namekey: org.apache.spark.rdd.RDD[(Int, String)] = MapPartitionsRDD[7] at keyBy at <console>:17

scala> namekey.collect
16/04/16 16:33:52 INFO SparkContext: Starting job: collect at <console>:20
16/04/16 16:33:52 INFO DAGScheduler: Got job 2 (collect at <console>:20) with 4 output partitions (allowLocal=false)
16/04/16 16:33:52 INFO DAGScheduler: Final stage: ResultStage 3(collect at <console>:20)
```



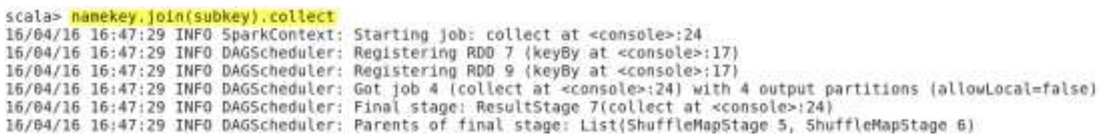
```
16/04/16 16:33:52 INFO Executor: Finished task 2.0 in stage 3.0 (TID 14). 764 bytes result sent to driver
16/04/16 16:33:52 INFO Executor: Finished task 3.0 in stage 3.0 (TID 15). 787 bytes result sent to driver
16/04/16 16:33:52 INFO TaskSetManager: Finished task 2.0 in stage 3.0 (TID 14) in 21 ms on localhost (3/4)
16/04/16 16:33:52 INFO TaskSetManager: Finished task 3.0 in stage 3.0 (TID 15) in 22 ms on localhost (4/4)
16/04/16 16:33:52 INFO TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have all completed, from pool
16/04/16 16:33:52 INFO DAGScheduler: ResultStage 3 (collect at <console>:20) finished in 0.032 s
16/04/16 16:33:52 INFO DAGScheduler: Job 2 finished: collect at <console>:20, took 0.104835 s
res2: Array[(Int, String)] = Array((3,kim), (5,kumar), (5,muthu), (3,tim), (3,lak), (4,vams))
```



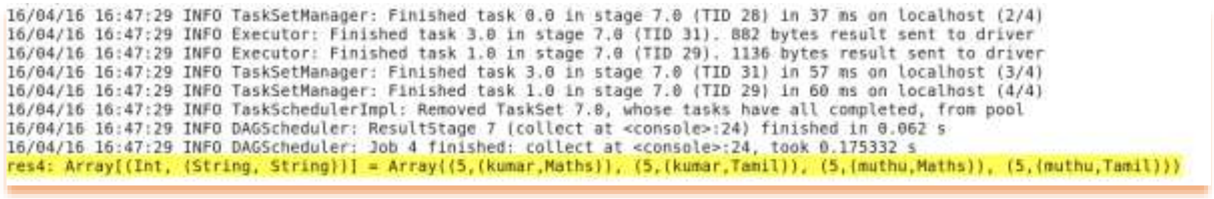
```
scala> val sub = sc.parallelize(List ("English","Maths","Tamil","Science"))
sub: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[8] at parallelize at <console>:15

scala> val subkey = sub.keyBy(_.length)
subkey: org.apache.spark.rdd.RDD[(Int, String)] = MapPartitionsRDD[9] at keyBy at <console>:17

scala> subkey.collect
16/04/16 16:46:20 INFO SparkContext: Starting job: collect at <console>:20
16/04/16 16:46:20 INFO DAGScheduler: Got job 3 (collect at <console>:20) with 4 output partitions (allowLocal=false)
16/04/16 16:46:20 INFO DAGScheduler: Final stage: ResultStage 4(collect at <console>:20)
16/04/16 16:46:20 INFO DAGScheduler: Parents of final stage: List()
```



```
16/04/16 16:46:20 INFO TaskSetManager: Finished task 2.0 in stage 4.0 (TID 18) in 17 ms on localhost (2/4)
16/04/16 16:46:20 INFO TaskSetManager: Finished task 3.0 in stage 4.0 (TID 19) in 17 ms on localhost (3/4)
16/04/16 16:46:20 INFO Executor: Finished task 1.0 in stage 4.0 (TID 17). 766 bytes result sent to driver
16/04/16 16:46:20 INFO TaskSetManager: Finished task 1.0 in stage 4.0 (TID 17) in 22 ms on localhost (4/4)
16/04/16 16:46:20 INFO DAGScheduler: ResultStage 4 (collect at <console>:20) finished in 0.019 s
16/04/16 16:46:20 INFO TaskSchedulerImpl: Removed TaskSet 4.0, whose tasks have all completed, from pool
16/04/16 16:46:20 INFO DAGScheduler: Job 3 finished: collect at <console>:20, took 0.035289 s
res3: Array[(Int, String)] = Array((7,English), (5,Maths), (5,Tamil), (7,Science))
```



```
scala> namekey.join(subkey).collect
16/04/16 16:47:29 INFO SparkContext: Starting job: collect at <console>:24
16/04/16 16:47:29 INFO DAGScheduler: Registering RDD 7 (keyBy at <console>:17)
16/04/16 16:47:29 INFO DAGScheduler: Registering RDD 9 (keyBy at <console>:17)
16/04/16 16:47:29 INFO DAGScheduler: Got job 4 (collect at <console>:24) with 4 output partitions (allowLocal=false)
16/04/16 16:47:29 INFO DAGScheduler: Final stage: ResultStage 7(collect at <console>:24)
16/04/16 16:47:29 INFO DAGScheduler: Parents of final stage: List(ShuffleMapStage 5, ShuffleMapStage 6)
```



```
16/04/16 16:47:29 INFO TaskSetManager: Finished task 0.0 in stage 7.0 (TID 28) in 37 ms on localhost (2/4)
16/04/16 16:47:29 INFO Executor: Finished task 3.0 in stage 7.0 (TID 31). 882 bytes result sent to driver
16/04/16 16:47:29 INFO Executor: Finished task 1.0 in stage 7.0 (TID 29). 1136 bytes result sent to driver
16/04/16 16:47:29 INFO TaskSetManager: Finished task 3.0 in stage 7.0 (TID 31) in 57 ms on localhost (3/4)
16/04/16 16:47:29 INFO TaskSetManager: Finished task 1.0 in stage 7.0 (TID 29) in 60 ms on localhost (4/4)
16/04/16 16:47:29 INFO TaskSchedulerImpl: Removed TaskSet 7.0, whose tasks have all completed, from pool
16/04/16 16:47:29 INFO DAGScheduler: ResultStage 7 (collect at <console>:24) finished in 0.062 s
16/04/16 16:47:29 INFO DAGScheduler: Job 4 finished: collect at <console>:24, took 0.175332 s
res4: Array[(Int, (String, String))] = Array((5,(kumar,Maths)), (5,(kumar,Tamil)), (5,(muthu,Maths)), (5,(muthu,Tamil)))
```

Example 17: Join

Inner Join

val names1 = sc.parallelize(List(“apple”, “mango”, “grapes”)).map(a => (a,1))

val names2 = sc.parallelize(List(“grapes”, “litchi”, “pears”)).map(a => (a,1))

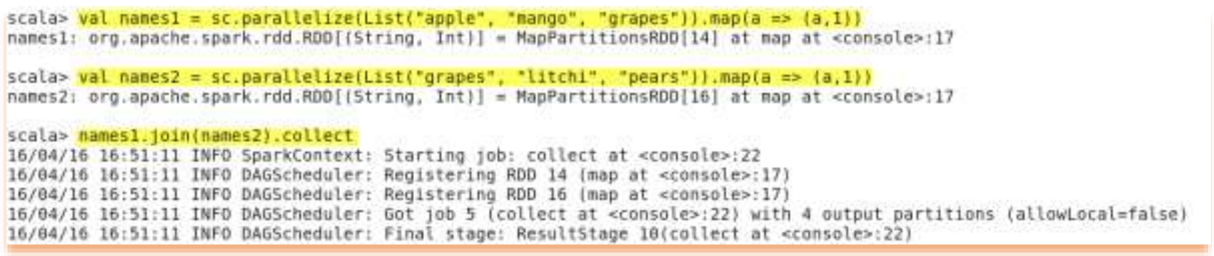
names1.join(names2).collect

leftOuterJoin

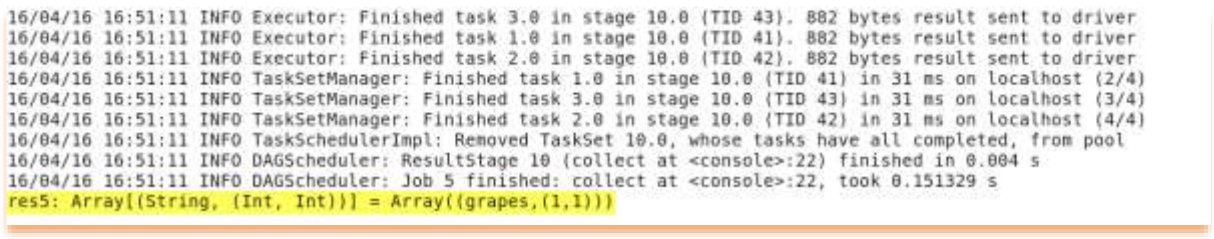
names1.leftOuterJoin(names2).collect

rightOuterJoin

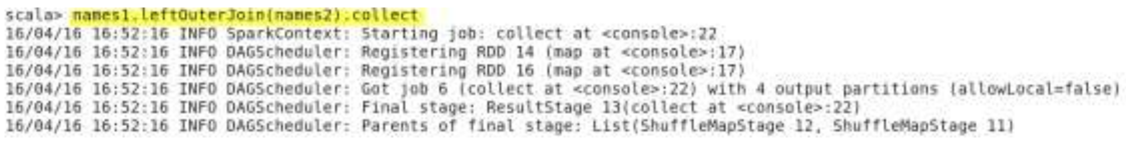
names1.rightOuterJoin(names2).collect



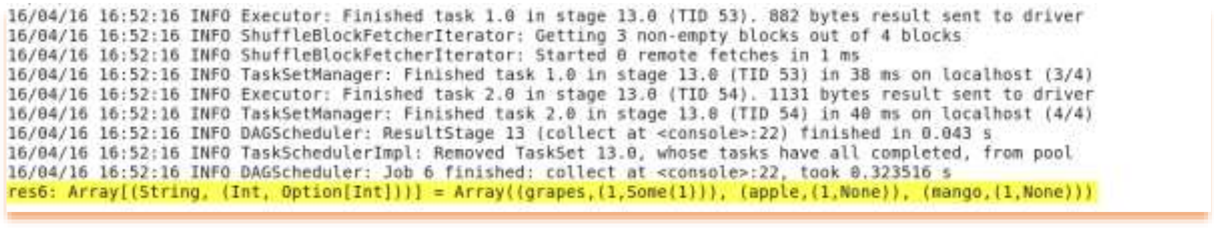
```
scala> val names1 = sc.parallelize(List("apple", "mango", "grapes")).map(a => (a,1))
names1: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[14] at map at <console>:17

scala> val names2 = sc.parallelize(List("grapes", "litchi", "pears")).map(a => (a,1))
names2: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[16] at map at <console>:17

scala> names1.join(names2).collect
16/04/16 16:51:11 INFO SparkContext: Starting job: collect at <console>:22
16/04/16 16:51:11 INFO DAGScheduler: Registering RDD 14 (map at <console>:17)
16/04/16 16:51:11 INFO DAGScheduler: Registering RDD 16 (map at <console>:17)
16/04/16 16:51:11 INFO DAGScheduler: Got job 5 (collect at <console>:22) with 4 output partitions (allowLocal=false)
16/04/16 16:51:11 INFO DAGScheduler: Final stage: ResultStage 10(collect at <console>:22)
```



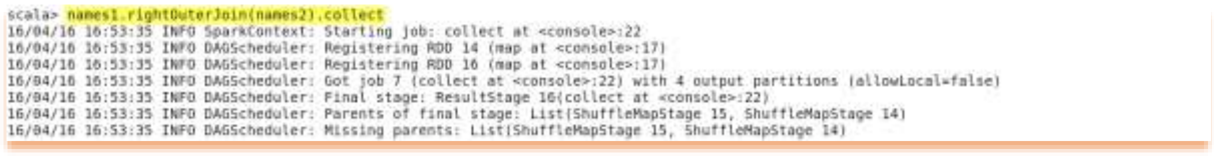
```
16/04/16 16:51:11 INFO Executor: Finished task 3.0 in stage 10.0 (TID 43). 882 bytes result sent to driver
16/04/16 16:51:11 INFO Executor: Finished task 1.0 in stage 10.0 (TID 41). 882 bytes result sent to driver
16/04/16 16:51:11 INFO Executor: Finished task 2.0 in stage 10.0 (TID 42). 882 bytes result sent to driver
16/04/16 16:51:11 INFO TaskSetManager: Finished task 1.0 in stage 10.0 (TID 41) in 31 ms on localhost (2/4)
16/04/16 16:51:11 INFO TaskSetManager: Finished task 3.0 in stage 10.0 (TID 43) in 31 ms on localhost (3/4)
16/04/16 16:51:11 INFO TaskSetManager: Finished task 2.0 in stage 10.0 (TID 42) in 31 ms on localhost (4/4)
16/04/16 16:51:11 INFO TaskSchedulerImpl: Removed TaskSet 10.0, whose tasks have all completed, from pool
16/04/16 16:51:11 INFO DAGScheduler: ResultStage 10 (collect at <console>:22) finished in 0.004 s
16/04/16 16:51:11 INFO DAGScheduler: Job 5 finished: collect at <console>:22, took 0.151329 s
res5: Array[(String, (Int, Int))] = Array((grapes,(1,1)))
```



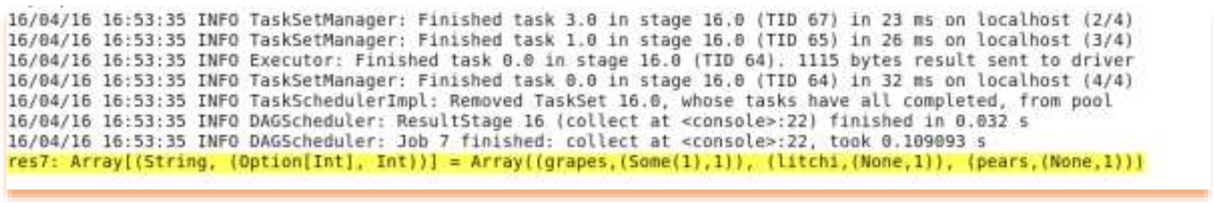
```
scala> names1.leftOuterJoin(names2).collect
16/04/16 16:52:16 INFO SparkContext: Starting job: collect at <console>:22
16/04/16 16:52:16 INFO DAGScheduler: Registering RDD 14 (map at <console>:17)
16/04/16 16:52:16 INFO DAGScheduler: Registering RDD 16 (map at <console>:17)
16/04/16 16:52:16 INFO DAGScheduler: Got job 6 (collect at <console>:22) with 4 output partitions (allowLocal=false)
16/04/16 16:52:16 INFO DAGScheduler: Final stage: ResultStage 13(collect at <console>:22)
16/04/16 16:52:16 INFO DAGScheduler: Parents of final stage: List(ShuffleMapStage 12, ShuffleMapStage 11)
```



```
16/04/16 16:52:16 INFO Executor: Finished task 1.0 in stage 13.0 (TID 53). 882 bytes result sent to driver
16/04/16 16:52:16 INFO ShuffleBlockFetcherIterator: Getting 3 non-empty blocks out of 4 blocks
16/04/16 16:52:16 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 1 ms
16/04/16 16:52:16 INFO TaskSetManager: Finished task 1.0 in stage 13.0 (TID 53) in 38 ms on localhost (3/4)
16/04/16 16:52:16 INFO Executor: Finished task 2.0 in stage 13.0 (TID 54). 1131 bytes result sent to driver
16/04/16 16:52:16 INFO TaskSetManager: Finished task 2.0 in stage 13.0 (TID 54) in 40 ms on localhost (4/4)
16/04/16 16:52:16 INFO DAGScheduler: ResultStage 13 (collect at <console>:22) finished in 0.043 s
16/04/16 16:52:16 INFO TaskSchedulerImpl: Removed TaskSet 13.0, whose tasks have all completed, from pool
16/04/16 16:52:16 INFO DAGScheduler: Job 6 finished: collect at <console>:22, took 0.323516 s
res6: Array[(String, (Int, Option[Int]))] = Array((grapes,(1,Some(1))), (apple,(1,None)), (mango,(1,None)))
```



```
scala> names1.rightOuterJoin(names2).collect
16/04/16 16:53:35 INFO SparkContext: Starting job: collect at <console>:22
16/04/16 16:53:35 INFO DAGScheduler: Registering RDD 14 (map at <console>:17)
16/04/16 16:53:35 INFO DAGScheduler: Registering RDD 16 (map at <console>:17)
16/04/16 16:53:35 INFO DAGScheduler: Got job 7 (collect at <console>:22) with 4 output partitions (allowLocal=false)
16/04/16 16:53:35 INFO DAGScheduler: Final stage: ResultStage 16(collect at <console>:22)
16/04/16 16:53:35 INFO DAGScheduler: Parents of final stage: List(ShuffleMapStage 15, ShuffleMapStage 14)
16/04/16 16:53:35 INFO DAGScheduler: Missing parents: List(ShuffleMapStage 15, ShuffleMapStage 14)
```



```
16/04/16 16:53:35 INFO TaskSetManager: Finished task 3.0 in stage 16.0 (TID 67) in 23 ms on localhost (2/4)
16/04/16 16:53:35 INFO TaskSetManager: Finished task 1.0 in stage 16.0 (TID 65) in 26 ms on localhost (3/4)
16/04/16 16:53:35 INFO Executor: Finished task 0.0 in stage 16.0 (TID 64). 1115 bytes result sent to driver
16/04/16 16:53:35 INFO TaskSetManager: Finished task 0.0 in stage 16.0 (TID 64) in 32 ms on localhost (4/4)
16/04/16 16:53:35 INFO TaskSchedulerImpl: Removed TaskSet 16.0, whose tasks have all completed, from pool
16/04/16 16:53:35 INFO DAGScheduler: ResultStage 16 (collect at <console>:22) finished in 0.032 s
16/04/16 16:53:35 INFO DAGScheduler: Job 7 finished: collect at <console>:22, took 0.109093 s
res7: Array[(String, (Option[Int], Int))] = Array((grapes,(Some(1),1)), (litchi,(None,1)), (pears,(None,1)))
```

Exercise 4: Operations On Multiple RDDs

Overview

In this lab, we will look at several transformations and examine the optimizations and visualise with DAG.

Builds on

Previous labs for the transformations we'll use.

Run time

30-40 minutes

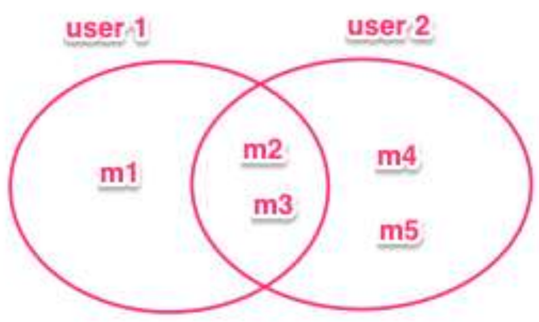
Meetups: Our data for this lab

For this lab, assume that we have users who are attending several meetups. We will start with two users, who are attending the following meetups:

User1 attends meetups: m1, m2 and m3. User2 attends meetups: m2, m3, m4 and m5.

Each user's meetups will be in separate RDDs, so you'll have two RDDs to work with. We'll analyze the data for the two users by performing operations over both RDDs. We'll also look at the Spark Shell UI to get an idea of how Spark is processing the data.

Operations that you work with will include union, intersection, distinct, and subtract .



***** Create the RDDs *****

```
scala> val u1 = sc.parallelize(List("m1", "m2", "m3"))
u1: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[0] at parallelize at
<console>:24


scala> val u2 = sc.parallelize(List("m2", "m3", "m4", "m5"))
```

```
u2: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[1] at parallelize at
<console>:24
```

-- Once you've done that, look at the Jobs tab of the UI - do you see anything?

You won't see any jobs yet. There have been no actions, so Spark is being lazy, and not doing anything yet.

***** Using your two RDDs, find meetups common to both users**

```
scala> val common = u1.intersection(u2)

common: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[7] at intersection at
<console>:28


scala> common.collect

res0: Array[String] = Array(m2, m3)
```

***** Find meetups attended by either user1 or user2.**

```
scala> val either = u1.union(u2).distinct()

either: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[11] at distinct at
<console>:28


scala> either.collect

res1: Array[String] = Array(m1, m2, m3, m4, m5)
```

***** Find meetups for each user that only one attended**

-- Find meetups that ONLY u1 attended (That is, u1 attended, but u2 did not.)

```
scala> val onlyU1 = u1.subtract(u1.intersection(u2))

onlyU1: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[21] at subtract at
<console>:28


scala> onlyU1.collect

res2: Array[String] = Array(m1)
```

-- Find meetups that ONLY u2 attended (That is, u2 attended, but u1 did not.)

```
scala> val onlyU2 = u2.subtract(u1.intersection(u2))

onlyU2: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[31] at subtract at
<console>:28


scala> onlyU2.collect

res3: Array[String] = Array(m5, m4)
```

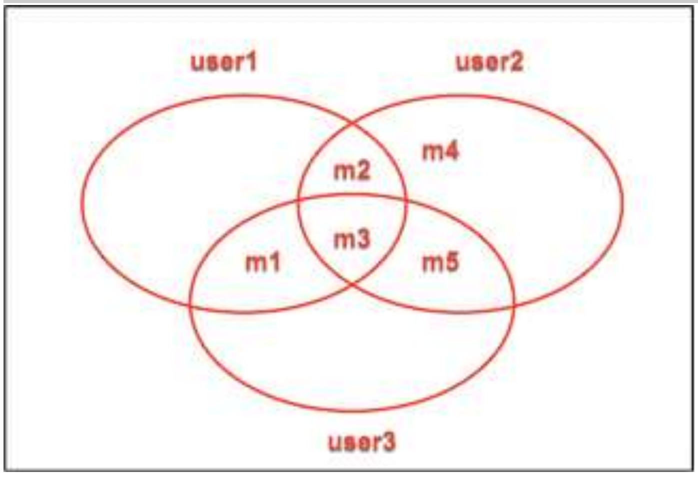
***** Find recommendations based on the requirements in the lab.**

*** A user should not be attending a meetup to be recommended.**

*** A meetup should be in attended by both other users to be recommended.**

Let's introduce user3

User3 attends meetups: m1, m3, and m5, as illustrated below.



Based on this, we would recommend the following to the users:
u1: m5
u2: m1
u3: m2

Tasks

- ➢ Create an RDD (u3) with the meetups it is attending (m1, m3, m5).
- ➢ Using your three RDDs, find the recommendations for u1 based on the rules above.
  - Look at the diagram above, and put into words how you would compute the recommendations.
  - Once you have a clear idea of how to do it, perform the RDD operations to

accomplish it in the Spark Shell.

- You'll need to use all 3 RDDs in your transformation for this. Look at the job DAG in the UI after you've done this.

-- Create u3

```
scala> val u3 = sc.parallelize(List("m1", "m3", "m5"))

u3: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[32] at parallelize at
<console>:24
```

-- Recommendations for u1:

```
scala> val forU1 = u2.intersection(u3).subtract(u1)

forU1: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[42] at subtract at
<console>:30


scala> forU1.collect

res4: Array[String] = Array(m5)
```

-- Recommendations for u2:

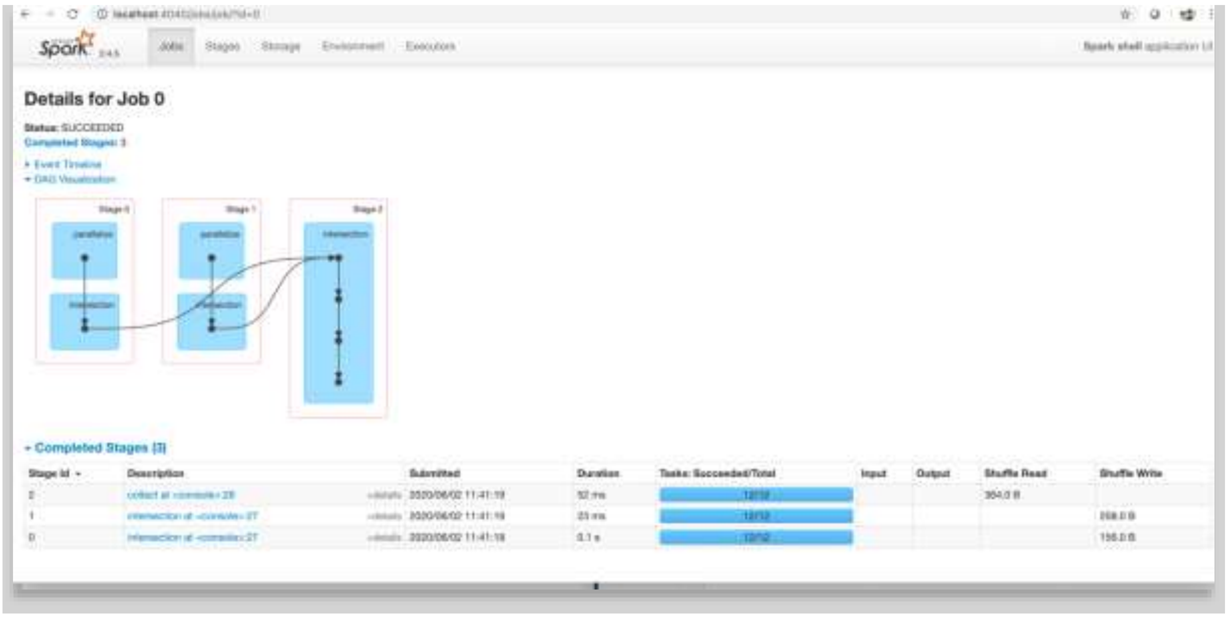
```
scala> val forU2 = u1.intersection(u3).subtract(u2)

forU2: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[52] at subtract at
<console>:30


scala> forU2.collect

res5: Array[String] = Array(m1)
```

-- Recommendations for u3:

```
scala> val forU3 = u1.intersection(u2).subtract(u3)

forU3: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[62] at subtract at
<console>:30


scala> forU3.collect

res6: Array[String] = Array(m2)
```

Discussions on What's Seen in the Spark Shell UI

Let's consider at some of the results you might have seen earlier in the UI.Here's the DAG from finding the meetups in common (an intersection of the two RDDs).



Details for Job 0

Status: SUCCEEDED
Completed Stages: 3

▸ Event Timeline
▾ DAG Visualization

▾ Completed Stages (3)

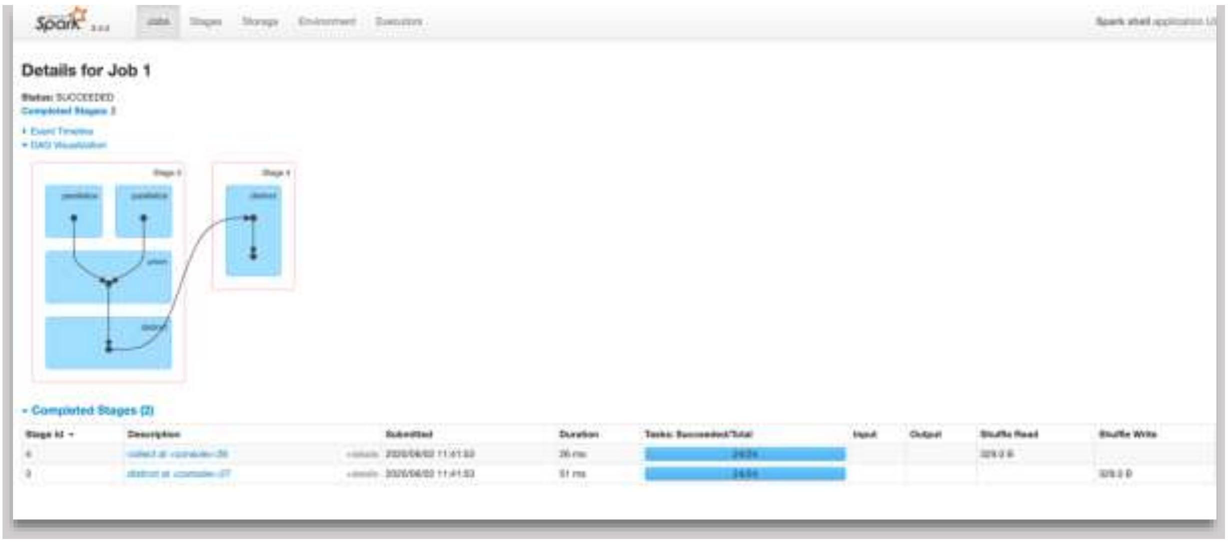
Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
2	collect at <console>:28	+details	2020/06/02 11:41:19	52 ms	12/12			364.0 B	
1	intersection at <console>:27	+details	2020/06/02 11:41:19	23 ms	12/12				208.0 B
0	intersection at <console>:27	+details	2020/06/02 11:41:18	0.1 s	12/12				156.0 B

What you're seeing is this.

- Each shaded blue box represents a user operation.
- A dot in the box represents an RDD created in the corresponding operations.
- Operations are grouped by stages (In a stage, operations on partitions are pipelined in the same task).
- You can click on any stage, to drill down into it for more detail.
- Parallelization of each RDD occurs in one stage (e.g. on one node, with local data)
- Some of the intersection can happen on one stage (using whatever data is local)
- Some of the intersection happens in another stage
  - Because it can no longer be done with local data - it involves data distributed over the cluster.
  - Data is **shuffled** for the intersection to be done (i.e. sent from one node to another).
  - Shuffling is expensive - we'll talk more about this later.

Here are the details on the stages (from the **Completed Stages** section on the same page). It gives details on the data that is shuffled.

Here's another DAG - from finding the meetups attended by **either** user (a union of the two RDDs).



Details for Job 1

Status: SUCCEEDED
Completed Stages: 2

▸ Event Timeline
▾ DAG Visualization

▾ Completed Stages (2)

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
4	collect at <console>:26 +details	2020/06/02 11:41:53	36 ms	24/24			329.0 B	
3	distinct at <console>:27 +details	2020/06/02 11:41:53	51 ms	24/24				329.0 B

What you're seeing is this.

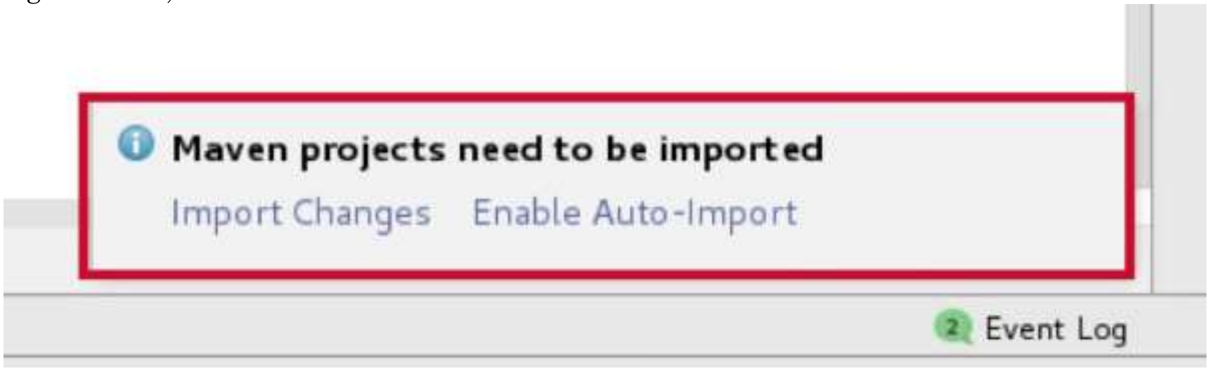
- ➢ Parallelization of each RDD occurs in one stage (e.g. on one node, with local data)
  - And yes, Spark has to parallelize again, because default is not to cache an RDD.
- ➢ The union happens in the same stage - It can all be done with local data
  - Part of the distinct has to be done in a separate stage - it also requires shuffling data

Exercise5: Building Spark Application using IntelliJ IDE

Overview
In this lab, we will look at several transformations and examine the optimizations that Catalyst performs.
Builds on
Previous labs for the transformations we'll use.
Run time
20-30 minutes

Goal: Create a scala project for Spark. Use maven for lib management.

1. Launch IntelliJ using the link on your lab environment desktop.
2. Click Open to open a project.
3. Locate and select the project's root folder (Lesson4, which you downloaded earlier) and click
4. OK. Wait while the project loads and the workspace is prepared.
5. If there's a "Maven projects need to be imported" popup message in the lower-right corner, click



6. When you have finished the code, build the JAR file which will be submitted to Spark. To do this
   with IntelliJ:
   a. Navigate to View > Tool Windows > Maven Projects. This opens a Maven Projects
   pane on the right side of the screen. You will see sfpdapp listed as a project:
7. Now add a new Scala object file " AirportsByLatitude" to project.

   Use Case: Create a Spark program to read the airport data from airports.text, find all the airports whose latitude are bigger than 40.
   Then output the airport's name and the airport's latitude to airports_by_latitude.
   Each row of the input file contains the following columns:

Airport ID, Name of airport, Main city served by airport, Country where airport is located, IATA/FAA code,
ICAO Code, Latitude, Longitude, Altitude, Timezone, DST, Timezone in Olson format

Sample output:
"St Anthony", 51.391944
"Tofino", 49.082222

```
import org.apache.spark.{SparkConf, SparkContext}


object AirportsByLatitude{
  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("airports").setMaster("local[2]")
    val sc = new SparkContext(conf)
    val airports = sc.textFile("<hdfs path>/airports.text")
    val airportsInUSA = airports.filter(line =>
line.split(Utils.COMMA_DELIMITER)(6).toFloat > 40)
    val airportsNameAndCityNames = airportsInUSA.map(line => {
      val splits = line.split(Utils.COMMA_DELIMITER)
      splits(1) + ", " + splits(6)
    })
    airportsNameAndCityNames.saveAsTextFile("<hdfs path>/airports_by_latitude.text")
  }
}
```

The code is already imported into IntelliJ in TrainingSpark workspace.

Execute the code, Right click -> Run as Scala Application. It has created the output folder with the desired output. You can go and verify the output.

Use Case:

Create a Spark program to read the airport data from airports.text,
find all the airports which are located in United States and output the airport's name and the city's name to out/airports_in_usa.

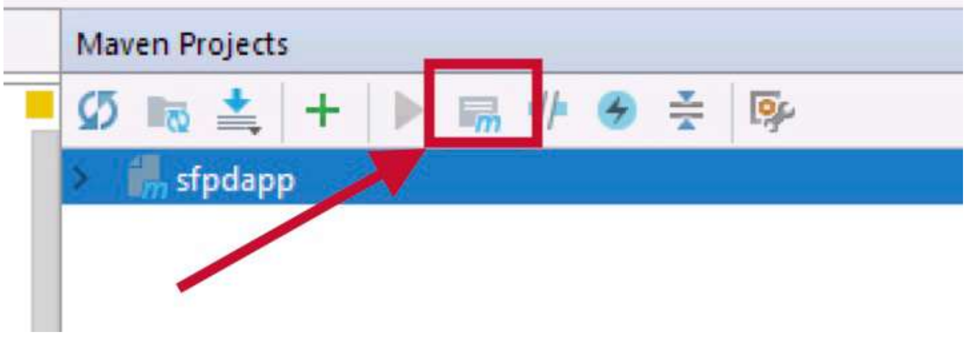
Each row of the input file contains the following columns:

Airport ID, Name of airport, Main city served by airport, Country where airport is located,IATA/FAA code, ICAO Code, Latitude, Longitude, Altitude, Timezone, DST, Timezone in Olson format

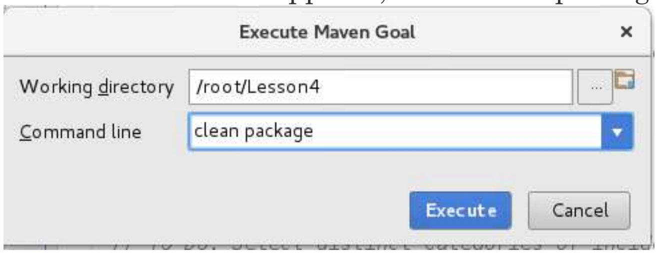
Sample output:

"Putnam County Airport", "Greencastle"
"Dowagiac Municipal Airport", "Dowagiac"

8. Click the Execute Maven Goal icon in the top bar of the Maven Projects window:



In the window that appears, enter clean package, and then click Execute:



Execute Maven Goal

Working directory: /root/Lesson4

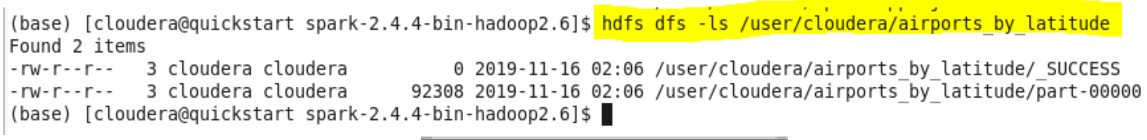
Command line: clean package

9. When it has finished downloading repositories and packaging the project, you will see a BUILD SUCCESS message in the status window. This may take a few minutes.
10. Open a terminal window on the host, and

11. Take this jar and submit using spark-submit command

```
cd /<path>/Lesson4/target

Navigate to the folder where Spark is installed.

$ bin/spark-submit --master yarn --class solution.AirportsByLatitude
/<path>/name of the jar
```

Next verify the output in the output directory of hdfs.



```
(base) [cloudera@quickstart spark-2.4.4-bin-hadoop2.6]$ hdfs dfs -ls /user/cloudera/airports_by_latitude
Found 2 items
-rw-r--r--   3 cloudera cloudera          0 2019-11-16 02:06 /user/cloudera/airports_by_latitude/_SUCCESS
-rw-r--r--   3 cloudera cloudera      92308 2019-11-16 02:06 /user/cloudera/airports_by_latitude/part-00000
(base) [cloudera@quickstart spark-2.4.4-bin-hadoop2.6]$ 
```

Exercise 6: Accumulators

Here we have the results of stack overflow annual salary survey for developers worldwide. This is a subset of the survey results that's open to CSV via an Excel. The first row is the header we are mostly interested in the third column which is the country of the developer and the 15th column which is the salary made a point.

But as you see we have some responses which don't have the salary midpoint records here. We want to answer several questions.

- First how many records do we have in the survey result.
- Second how many records are missing the salary middle point third.
- How many records are from Canada.

But we want to answer all those questions by passing over all the data only once.

Let us create a Scala class as StackOverFlowSurvey in the package com.sparkTutorial.advanced.accumulator with the below code.

```
package com.sparkTutorial.advanced.accumulator
import com.sparkTutorial.commons.Utils
import org.apache.log4j.{Level, Logger}
import org.apache.spark.{SparkConf, SparkContext}

object StackOverFlowSurvey {

  def main(args: Array[String]) {
    Logger.getLogger("org").setLevel(Level.ERROR)
    val conf = new
SparkConf().setAppName("StackOverFlowSurvey").setMaster("local[1]")
    val sparkContext = new SparkContext(conf)

    val total = sparkContext.longAccumulator
    val missingSalaryMidPoint = sparkContext.longAccumulator

    val responseRDD = sparkContext.textFile("in/2016-stack-overflow-
survey-responses.csv")

    val responseFromCanada = responseRDD.filter(response => {
      val splits = response.split(Utils.COMMA_DELIMITER, -1)
      total.add(1)

      if (splits(14).isEmpty) {
        missingSalaryMidPoint.add(1)
      }

      splits(2) == "Canada"
    })

    println("Count of responses from Canada: " +
responseFromCanada.count())
    println("Total count of responses: " + total.value)
    println("Count of responses missing salary middle point: " +
missingSalaryMidPoint.value)
  }
}
```

Exercise 7: Broadcast variables

Here we have the maker space data across the UK. The first row is the header. This data set includes the name of the maker space email address postcard number of visitors etc. We want to answer the following question.

- How are those makers space distributed across different regions in the UK.

However, we only have the postcode for each maker space.

We don't know the region of each maker's space to answer the above question.

We need to introduce another dataset here.

We also have the UK postcode data for each postcode prefix we can find out which region it belongs to.

Combining those two data sets we should be able to answer the question how are those makers spaces distributed across different regions in the UK.

Let us create a Scala class as UkMakerSpaces in the package "com.sparkTutorial.advanced.broadcast" and paste the below code:

```
package com.sparkTutorial.advanced.broadcast

import com.sparkTutorial.commons.Utils
import org.apache.log4j.{Level, Logger}
import org.apache.spark.{SparkConf, SparkContext}

import scala.io.Source

object UkMakerSpaces {

  def main(args: Array[String]) {
    Logger.getLogger("org").setLevel(Level.ERROR)
    val conf = new
SparkConf().setAppName("UkMakerSpaces").setMaster("local[1]")
    val sparkContext = new SparkContext(conf)

    val postCodeMap = sparkContext.broadcast(loadPostCodeMap())

    val makerSpaceRdd = sparkContext.textFile("in/uk-makerspaces-
identifiable-data.csv")

    val regions = makerSpaceRdd
      .filter(line => line.split(Utils.COMMA_DELIMITER, -1)(0) !=
"Timestamp")
      .filter(line => getPostPrefix(line).isDefined)
      .map(line =>
postCodeMap.value.getOrElse(getPostPrefix(line).get, "Unknown"))

    for ((region, count) <- regions.countByValue()) println(region +
" : " + count)
```

```
  }

  def getPostPrefix(line: String): Option[String] = {
    val splits = line.split(Utils.COMMA_DELIMITER, -1)
    val postcode = splits(4)
    if (postcode.isEmpty) None else Some(postcode.split(" ")(0))
  }

  def loadPostCodeMap(): Map[String, String] = {
    Source.fromFile("in/uk-postcode.csv").getLines.map(line => {
      val splits = line.split(Utils.COMMA_DELIMITER, -1)
      splits(0) -> splits(7)
    }).toMap
  }
}
```

Exercise 8: Dataframe, Datasets and Spark SQL

Two ways to create Spark Dataframes:

Creating DataFrames from Existing RDD:

1. Infer schema by Reflection
   a. Convert RDD containing case classes.
   b. Use when schema is known.

2. Construct schema programmatically
   a. Schema is dynamic
   b. Number of fields in case class is more than 22 fields.

TWO WAYS TO DEFINE A SCHEMA

1. Define Schema Programmatically

```
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions.{col, expr}
val jsonFile = “file:///home/cloudera/Desktop/SW/in/Data/blogs.json”
  // Define our schema programmatically
  val schema = StructType(Array(StructField("Id", IntegerType, false),
    StructField("First", StringType, false),
    StructField("Last", StringType, false),
    StructField("Url", StringType, false),
    StructField("Published", StringType, false),
    StructField("Hits", IntegerType, false),
    StructField("Campaigns", ArrayType(StringType), false)))
/Create a DataFrame by reading from the JSON file a predefined Schema
   val blogsDF = spark.read.schema(schema).json(jsonFile)
   //show the DataFrame schema as output
   blogsDF.show(truncate = false)
   // print the schemas
   print(blogsDF.printSchema)

   print(blogsDF.schema)

   // Show columns and expressions

   blogsDF.select(expr("Hits") * 2).show(2)

   blogsDF.select(col("Hits") * 2).show(2)

   blogsDF.select(expr("Hits * 2")).show(2)

  // show heavy hitters

  blogsDF.withColumn("Big Hitters", (expr("Hits > 10000"))).show()

// Concatenate three columns, create a new column, and show the

// newly created concatenated column

blogsDF.withColumn("AuthorsId",        (concat(expr("First"),        expr("Last"),
expr("Id")))).select(col("AuthorsId")).show(4)


// Sort by column "Id" in descending order

blogsDF.sort(col("Id").desc).show()

blogsDF.sort($"Id".desc).show()

// In Scala to save as a Parquet file

val parquetPath = <path>

blogsDF.write.format("parquet").save(parquetPath)
```

Note: the expressions blogs_df.sort(col("Id").desc) and blogsDF.sort($"Id".desc) are identical. They both sort the DataFrame column named Id in descending order: one uses an explicit function, col("Id"), to return a Column object, while the other uses $ before the name of the column, which is a function in Spark that converts column named Id to a Column.

Assignment 1 : let's read a large CSV file containing data on San Francisco Fire Department calls. we will define a schema for this file and use the DataFrameReader class and its methods to tell Spark what to do. Because this file contains 28 columns and over 4,380,660 records,2 it's more efficient to define a schema than have Spark infer it.

Hint:

```
val sfFireFile="/path/sf-fire-calls.csv"
val fireDF = spark.read.schema(fireSchema)
  .option("header", "true")
  .csv(sfFireFile)
```

- What were all the different types of fire calls in 2018?
- What months within the year 2018 saw the highest number of fire calls?
- Which neighborhood in San Francisco generated the most fire calls in 2018?
- Which neighborhoods had the worst response times to fire calls in 2018?
- Which week in the year in 2018 had the most fire calls?
- Is there a correlation between neighborhood, zip code, and number of fire calls?
- How can we use Parquet files or SQL tables to store this data and read it back?

2. Infer schema by Reflection

let's look at a collection of readings from Internet of Things (IoT) devices in a JSON file (we use this file in the end-to-end example later in this section).

Let us create Case class for the file

```
case class DeviceIoTData (battery_level: Long, c02_level: Long,

cca2: String, cca3: String, cn: String, device_id: Long,

device_name: String, humidity: Long, ip: String, latitude: Double,

lcd: String, longitude: Double, scale:String, temp: Long,

timestamp: Long)


val ds =
spark.read.json("file:///home/cloudera/Desktop/SW/in/Data/iot_devices.json").as[DeviceIoTData]

val filterTempDS = ds.filter({d => {d.temp > 30 && d.humidity > 70})

filterTempDS.show(5, false)


// In Scala
```

```
case class DeviceTempByCountry(temp: Long, device_name: String, device_id: Long,
  cca3: String)
val dsTemp = ds
  .filter(d => {d.temp > 25})
  .map(d => (d.temp, d.device_name, d.device_id, d.cca3))
  .toDF("temp", "device_name", "device_id", "cca3")
  .as[DeviceTempByCountry]
dsTemp.show(5, false)
```

Alternatively, you could express the same query using column names and then cast to a Dataset[DeviceTempByCountry]:

```
// In Scala
val dsTemp2 = ds
  .select($"temp", $"device_name", $"device_id", $"device_id", $"cca3")
  .where("temp > 25")
  .as[DeviceTempByCountry]
```

Assignment 2: Using the Dataset API, try to execute to do the following:

- Detect failing devices with battery levels below a threshold.
- Identify offending countries with high levels of CO2 emissions.
- Compute the min and max values for temperature, battery level, CO2, and humidity.
- Sort and group by average temperature, CO2, humidity, and country.

Using Spark SQL with queries

We will here work on the Airline On-Time Performance and Causes of Flight Delays data set, which contains data on US flights including date, delay, distance, origin, and destination. It's available as a CSV file with over a million records. Using a schema, we'll read the data into a DataFrame and register the DataFrame as a temporary view (more on temporary views shortly) so we can query it with SQL.

```
// Path to data set

val csvFile="/home/cloudera/Desktop/SW/in/Data/departuredelays.csv"



// Read and create a temporary view

// Infer schema (note that for larger files you may want to specify the schema)

val df = spark.read.format("csv").option("inferSchema", "true").option("header",
"true").load(csvFile)

// Create a temporary view

df.createOrReplaceTempView("us_delay_flights_tbl")
```

Now that we have a temporary view, we can issue SQL queries using Spark SQL. These queries are no different from those you might issue against a SQL table in, say, a MySQL or PostgreSQL database. The point here is to show that Spark SQL offers an ANSI:2003–compliant SQL interface, and to demonstrate the interoperability between SQL and DataFrames.

The US flight delays data set has five columns:

- The `date` column contains a string like `02190925`. When converted, this maps to `02-19 09:25 am`.
- The `delay` column gives the delay in minutes between the scheduled and actual departure times. Early departures show negative numbers.
- The `distance` column gives the distance in miles from the origin airport to the destination airport.
- The `origin` column contains the origin IATA airport code.
- The `destination` column contains the destination IATA airport code.

With that in mind, let's try some example queries against this data set.

➢ First, we'll find all flights whose distance is greater than 1,000 miles:

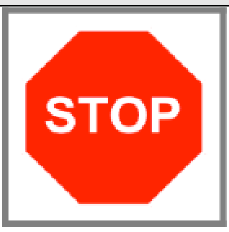
```
spark.sql("""SELECT  distance,  origin,  destination  FROM  us_delay_flights_tbl
WHERE distance > 1000 ORDER BY distance DESC""").show(10)
```

- Next, we'll find all flights between San Francisco (SFO) and Chicago (ORD) with at least a two-hour delay:

```
spark.sql("""SELECT date, delay, origin, destination FROM us_delay_flights_tbl
WHERE delay > 120 AND ORIGIN = 'SFO' AND DESTINATION = 'ORD'
ORDER by delay DESC""").show(10)
```

- **Assignment**: It seems there were many significantly delayed flights between these two cities, on different dates. (As an exercise, convert the date column into a readable format and find the days or months when these delays were most common. Were the delays related to winter months or holidays?)
- Let's try a more complicated query where we use the CASE clause in SQL. In the following example, we want to label all US flights, regardless of origin and destination, with an indication of the delays they experienced: Very Long Delays (> 6 hours), Long Delays (2–6 hours), etc. We'll add these human-readable labels in a new column called Flight_Delays:

```
spark.sql("""SELECT delay, origin, destination,
          CASE
             WHEN delay > 360 THEN 'Very Long Delays'
             WHEN delay > 120 AND delay < 360 THEN 'Long Delays'
             WHEN delay > 60 AND delay < 120 THEN 'Short Delays'
             WHEN delay > 0 and delay < 60  THEN  'Tolerable Delays'
             WHEN delay = 0 THEN 'No Delays'
             ELSE 'Early'
           END AS Flight_Delays
           FROM us_delay_flights_tbl
           ORDER BY origin, delay DESC""").show(10)
```



Overview

In this lab, we will look at several transformations and optimizations on Dataframes and Datasets.

Builds on

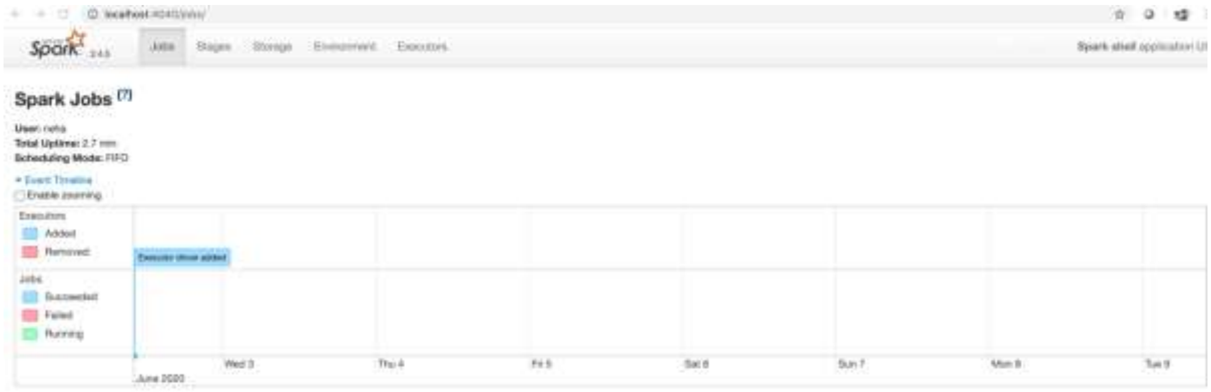
Previous labs for the transformations we'll use.
Run time
20-30 minutes

➢ Explore the Spark shell UI Tasks

Open the FireFox browser (In cloudera dekstop, **Applications Menu | FireFox**), and go to the launch **Spark_UI (localhost:4040)**

In the page that opens, look for the application

That should open the Spark UI Below is a sample screen shot of the UI



Spark Jobs (?)

User: neha
Total Uptime: 2.7 min
Scheduling Mode: FIFO

```
                         version 2.4.5

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

scala>
```

NOTE: In all screen shots, you might see differences in job ids, stage ids and other details that depend on the exact state of your Spark shell. This is because the shell used for the screen shots is likely to have executed a different overall sequence of transformations than your shell since it was started. This is not important - the important details should be easily recognizable.

<u>Lab: 10.1</u>

Spark SQL Basics

Overview

In this lab, we will use some of the basic functionality of Spark SQL, including:

Read and write data files in varying formats Create DataFrames

Run time

15-20 minutes

Lab Preparation

Tasks

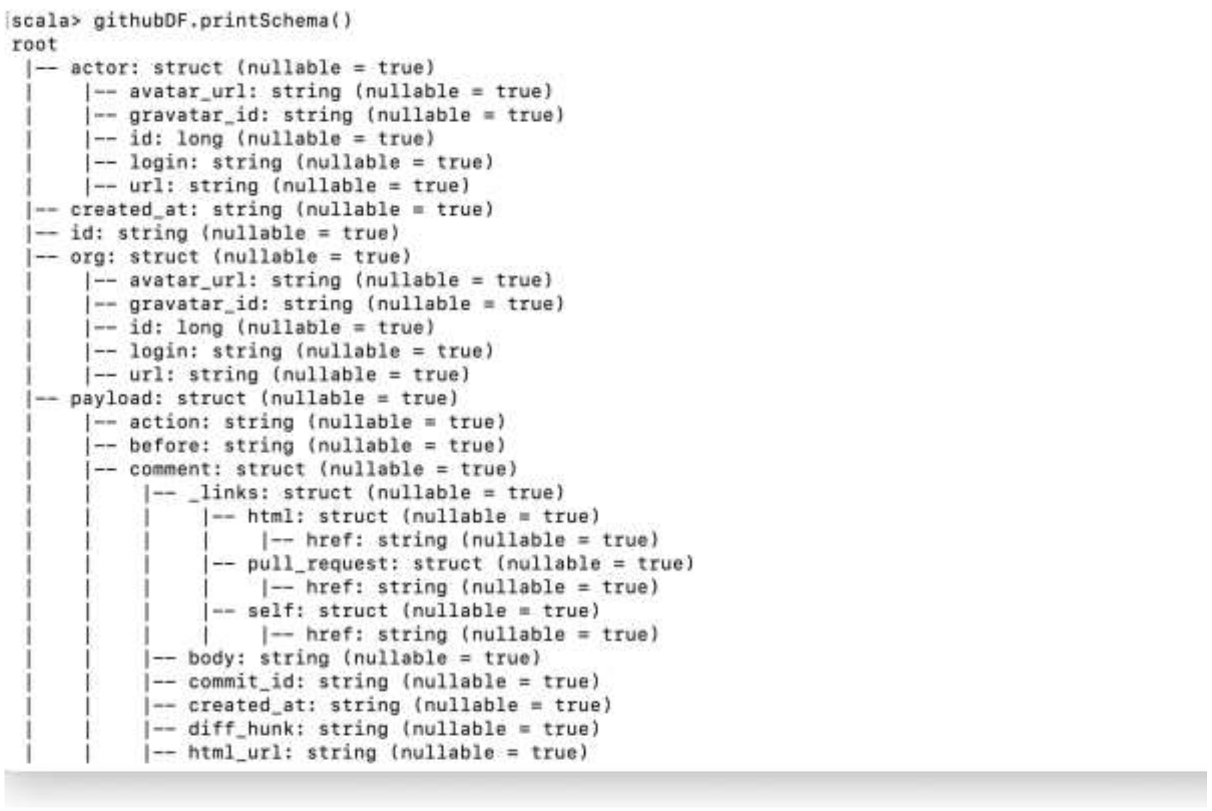
Load data from the following files, if you haven't already

- *people.json* as JSON
- *wiki-pageviews.txt* as text
- *github.json* as JSON

```
//Scala Code
***
*** Load Text and JSON Data ***
***

Read people.json
> val folksDF=spark.read.json("data/people.json")

Read wiki-pageviews.txt
> var viewsDF = spark.read.text("data/wiki-pageviews.txt")

Display some of the data
> folksDF.limit(5).show
> viewsDF.limit(5).show

***
*** Read More Complex Data ***
***
Read/display data in github.json

> val githubDF=spark.read.json("data/github.json")
> githubDF.limit(5).show

***
*** Write Data ***
***
```

```
Write out folksDF as parquet file
> folksDF.write.parquet("data/people.parquet")

Write out folksDF as a CSV file
> folksDF.coalesce(1).write.option("header", true).csv("data/people.csv")
```

View the Schema



```
scala> githubDF.printSchema()
root
 |-- actor: struct (nullable = true)
 |    |-- avatar_url: string (nullable = true)
 |    |-- gravatar_id: string (nullable = true)
 |    |-- id: long (nullable = true)
 |    |-- login: string (nullable = true)
 |    |-- url: string (nullable = true)
 |-- created_at: string (nullable = true)
 |-- id: string (nullable = true)
 |-- org: struct (nullable = true)
 |    |-- avatar_url: string (nullable = true)
 |    |-- gravatar_id: string (nullable = true)
 |    |-- id: long (nullable = true)
 |    |-- login: string (nullable = true)
 |    |-- url: string (nullable = true)
 |-- payload: struct (nullable = true)
 |    |-- action: string (nullable = true)
 |    |-- before: string (nullable = true)
 |    |-- comment: struct (nullable = true)
 |    |    |-- _links: struct (nullable = true)
 |    |    |    |-- html: struct (nullable = true)
 |    |    |    |    |-- href: string (nullable = true)
 |    |    |    |-- pull_request: struct (nullable = true)
 |    |    |    |    |-- href: string (nullable = true)
 |    |    |    |-- self: struct (nullable = true)
 |    |    |    |    |-- href: string (nullable = true)
 |    |    |-- body: string (nullable = true)
 |    |    |-- commit_id: string (nullable = true)
 |    |    |-- created_at: string (nullable = true)
 |    |    |-- diff_hunk: string (nullable = true)
 |    |    |-- html_url: string (nullable = true)
```

The githubDF schema is quite complex. We'll look at some ways of dealing with it soon.

<u>Declare a Schema Explicitly Tasks</u>

Declare a schema for folksDF. We illustrate this below. We also supply this code in a below snippet - *peopleSchema.txt* :

The file is stored in the Datasets.

```
import org.apache.spark.sql.types._
```

```
val mySchema = (new StructType).add("name", StringType).add("gender",
StringType).add("age", IntegerType)

Read data/people.json again, but this time supply the schema you declared.

> var folksDF = spark.read.schema(mySchema).json("data/people.json")
```

```
# Python

from pyspark.sql.types import *


mySchema =
StructType().add("name",StringType()).add("gender",StringType()).add("age",
IntegerType())

folksDF = spark.read.schema(mySchema).json("data/people.json")
```

<u>Lab 10.2</u>

DataFrame Transformations

Overview

In this lab, we will perform a number of transformations on dataframes. This will provide exposure to the API, and some experience in using it.

We will cover a number of common techniques, but will not attempt to cover all of the API, or try to illustrate every possible technique. The API is too large for that.

We'll be working at the Spark shell for all of these.

Run time

30-40 minutes

Assignment:

- Display the data.Filter on age and display the data.
- Filter on gender and display the data.
- Count how many "F" and "M" items there are.
- **[Optional]** Find the oldest person with gender "F".
    - Use SQL to write this query - e.g. spark.sql(" ... ")
    - Remember to register a temporary table.
    - Use a subquery to find the maximum age.

Solution:

***

*** Simple Transformations ***

***

Read people.json if you haven't already

```
> val folksDF=spark.read.json("data/people.json")
```

Display the data.

```
> folksDF.show
```

Filter on age and display the data

Filter on gender and display the data

```
> folksDF.filter('age>25).show // With age greater than 25

+---+------+----+

|age|gender|name|

+---+------+----+

| 35|     M|John|

| 40|     F|Jane|

| 52|     F| Sue|

+---+------+----+
```

```
> folksDF.filter('age>25 && 'age<50).show  // With age greater than 25 and less than 50

+---+------+----+

|age|gender|name|

+---+------+----+

| 35|     M|John|

| 40|     F|Jane|

+---+------+----+
> folksDF.filter('gender === "F").show
+---+------+----+
|age|gender|name|
+---+------+----+
| 40|     F|Jane|
| 52|     F| Sue|
+---+------+----+
```

Count how many "F" and "M" items there are.

```
> folksDF.groupBy('gender).count.show
+------+-----+
|gender|count|
+------+-----+
|     F|    2|
|     M|    2|
+------+-----+
```

[Optional] Find the oldest person with gender "F"

```
> folksDF.createOrReplaceTempView("people")
> spark.sql("SELECT name, age FROM people WHERE gender = 'F' and age =
(select max(age) FROM people WHERE gender='F')").show
+----+---+
|name|age|
+----+---+
| Sue| 52|
+----+---+
```

Working with More Complex Data

Tasks

- Read *github.json* if you haven't already

```
> val githubDF=spark.read.json("data/github.json")
```

- Look at the schema and 5 sample rows (limit(5).show) again.
- Look at the githubDF schema and a few rows (limit().show) again.
- Select the login value of the actor and display it.
- Find out how many unique logins there are in the data.
- Each row in this data, contains a type column. Display all the unique values for the 'type' column.
- Determine how may rows have a type of CreateEvent

```
> githubDF.limit(5).show


Select the actor column and view it and its schema

> githubDF.limit(5).select('actor).show(false)

> githubDF.select('actor).printSchema

Select the login value of the actor and display it

> actorDF.select("actor.login").limit(5).show


Find out how many unique logins there are in the data.

> actorDF.select("actor.login").dropDuplicates.count
```

Retrieve all the unique values for the 'type' column.

```
> gitHubDF.dropDuplicates("type").count

> gitHubDF.dropDuplicates("type").show
```

Working with Text Data

Tasks

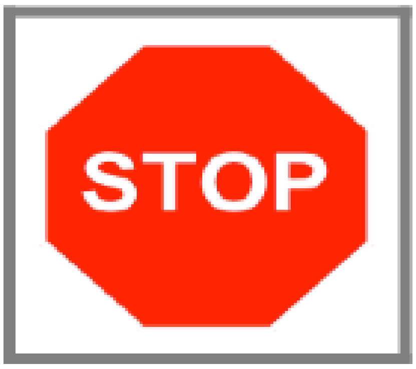
- Read *wiki-pageviews.txt* if you haven't already
- Show 5 rows from this data so you can examine it.

```
 val viewsDF=spark.read.text("<path>/wiki-pageviews.txt")

viewsDF.limit(5).show
```

- We can see that each row contains a single string.
  - Unlike JSON data, there is no automatic application of a schema.
  - This is a cumbersome format to work with, and we'll see ways to transform it into an easier to use format later.

Summary

We've practiced using some of the common transformations that are available for dataframes. We'll continue using this data to delve into Spark's capabilities.



Lab 10.3: The Dataset Typed API

Overview

In this lab, we will work with the Dataset typed API which provides compile-time type safety.

Continue working in your spark shell in this lab as previously. You'll work with some of the dataframes created in the previous lab.

What About Python?

Python does NOT have Dataset . It is a dynamically typed language, and thus the strongly typed Dataset makes no sense for it.

Run time

30-40 minutes

Music Data File

We've provided a small and simple JSON data file (***<path>/data/music.json***) containing data about music items. We'll use it to do our Dataset work in this lab. It has four pieces of data in it:

- title: Title of the item.
- artist: Artist who released the item.
- price: Price of the item
- category: The music category (e.g. Pop)

Tasks

- Create a DataFrame by reading in the music data in ***data/music.json***
- View the schema of this DataFrame.
- Display the data in the DataFrame (show).

Simple Dataset Usage Tasks

- Define a MusicItem case class suitable for our *music.json* data. Using this class, create a typed Dataset from the musicDF dataframe created
- with this data in the previous lab.
- What type is musicDS? Display the data in the dataset.

Solution:

```
***
*** Music Data File ***
***

val musicDF=spark.read.json("data/music.json")

> musicDF.schema

> musicDF.show

***
*** Simple DataSet Usage ***
***

Define a MusicItem case class suitable for our music.json data.
> case class MusicItem (title: String, artist: String, category: String, price: Double)

Using this class, create a typed DataSet from the musicDF dataframe.
> val musicDS=musicDF.as[MusicItem]

What type is folksDS?
DataSet of MusicItem, as shown below.
> musicDS

Display the data in the dataset.
> musicDS.show
```

Compare: DataFrame vs. Dataset

We'll perform some operations on our DataFrame and Dataset representations, to get a feel of the difference.

Tasks

- ➢ Filter on category
  - Using musicDF get all the items in the "Pop" category.
  - Using musicDS get all the items in the "Pop" category.
- ➢ Get lowest priced item in each category.
  - You'll need to group, then do an aggregation.
  - Using musicDF and untyped transformations, get the lowest price item in a category.
  - Using musicDS and typed transformations, get the lowest price item in a category.
- ➢ Transform data so that the price is reduced 10%. (You can do this by multiplying it by 0.9).
  - Using musicDF (the DataFrame) transform it to a DataFrame where the price is reduced by 10%.
    - o Hint use a select and a literal to create the new price values.

  - Using musicDS (the Dataset) transform it to a Dataset where the price is doubled.
    - o Use map()
    - o Make sure you know how your case class is defined (i.e. what order the
    - o fields appear).

- ➢ Make a mistake - watch how errors are caught.
  - Using musicDF (the DataFrame) make a mistake in transforming it when you modify the price.
    - o Try multiplying the price by a literal string - e.g. lit("A")
    - o What happens?
  - Using musicDS (the Dataset) make a mistake in transforming when you modify the price.
    - o Try multiplying the price by a string (e.g. "A")
    - o What happens?

Solution:

```
***
*** Compare: DataFrame vs. DataSet ***
***
```

```
-- Filter on category.
> musicDF.filter('category === "Pop").show

> musicDS.filter(mi => mi.category == "Pop").show

-- Using musicDF and untyped transformations, get the lowest price item in a category.

> musicDF.groupBy('category).min("price").show

>musicDS.groupByKey(mi=>
mi.category).agg(min('price).as[Double]).show


- Transform data so that the price is reduced 10%. (You can do this by multiplying it by 0.9).

// DataFrame
> musicDF.select('title, 'artist, 'category, 'price*lit(0.9)).show

-- Make a mistake - watch how errors are caught.

// DataFrame - error not caught - get erroneous results.

a> musicDF.select('title, 'artist, 'category, 'price*lit("A")).show


// DataSet - error caught at parsing time.
> musicDS.map (mi => MusicItem(mi.title, mi.artist, mi.category, mi.price*"A")).show
 > musicDS.map (mi => MusicItem(mi.title, mi.artist, mi.category, mi.price*"A")).show
```

Summary

We've practiced some fairly simple transformations with Datasets. We've also done the same thing with DataFrames to get a feel of the di!erence.

DataFrames are a little easier to program to. However, as we saw, Datasets can catch errors earlier than DataFrames. This can be important when debugging and maintaining a large system.

Lab 10.4: Splitting Text Data

Overview
In this lab, we'll work with text data. The data is regularly structured, but since it's in text format, Spark can't deduce the structure on its own.
We'll use some of the DataFrame tools to create a DataFrame with a schema that's easy to use for querying the data.
The data we'll use contains page view data from Wikimedia.

Run time
30-40 minutes

Wikimedia PageView Data File
We provide a data file (*<path>/data/wiki-pageviews.txt*) that contains a dump of pageview data for many projects under the Wikimedia umbrella.

The data file has lines containing four fields.

1. Domain.project (e.g. "en.b")
2. Page name (e.g. "AP_Biology/Evolution")
3. Page view count (e.g. 1)
4. Total response size in bytes (e.g. 10662 - but for this particular dump, value is always 0).
   The data is in simple text format, so when we read it in, we get a dataframe with a single column - a string containing all the data in each row. This is cumbersome to work with. In this lab, we'll apply a better schema to this data.

- **Tasks**Create a DataFrame by reading in the page view data in **spark-labs/data/wiki-pageviews.txt**.

- Once you've created it, view a few lines to see the format of the data.
    - You'll see that you have one line of input per dataframe row.

<u>**Split the Lines**</u>

Our first step in creating an easier to use schema is splitting each row into separate columns. We'll use the split() function defined in the Spark SQL funtions. We've used this in our word count examples in the main manual.

Tasks

- Create a dataframe by splitting each line up.
    - Use split() to split on a whitespace (pattern of "\\s+")

Python: You'll need to import from the functions module as shown in the Python example below

```
//Python
> from pyspark.sql.functions import *
> splitViewsDF = // ...
```

<u>**Create a Better Schema**</u>

We'll create a dataframe with an easier-to-use schema containing the following columns which align with the data in the views file.

- domain: String - The domain.project data.
- pageName: String - The page name.
- viewCount: Integer - The view count.
- size: Long - The response size (always 0 in this file, but we'll keep it in our dataframe).

Execute the below queries one after the other

```
***
*** Wikimedia PageView Data File ***
***

Read the data file, and show some lines
> val viewsDF=spark.read.text("data/wiki-pageviews.txt")
> viewsDF.limit(5).show
```

****
***** Split the Lines *****
****

Split on whitespace, show some lines, show the schema

```
> val splitViewsDF = viewsDF.select(split('value, "\\s+").as("splitLine"))
> splitViewsDF.limit(5).show(false)

> splitViewsDF.printSchema
```

****
***** Create a Better Schema *****
****

Create a dataframe with a better schema, view the schema, and view some data.

```
>val viewsWithSchemaDF =
splitViewsDF.select('splitLine(0).as("domain"),
'splitLine(1).as("pageName"),  'splitLine(2).cast("integer").as("viewCount"),
'splitLine(3).cast("long").as("size"))


> viewsWithSchemaDF.printSchema
> viewsWithSchemaDF.limit(5).show
```

***
*** Try Some Queries ***
***

Find rows where the viewCount > 500, and display 5 of them

```
> viewsWithSchemaDF.filter('viewCount>500).limit(5).show
```

Same query as above with domain of "en" - note the triple = for equality

```
>viewsWithSchemaDF.filter('domain==="en").filter('viewCount>500).limit
(5).show
```

Find the 5 rows with the largest viewCount and a domain of "en"

```
>viewsWithSchemaDF.filter('domain==="en").orderBy('viewcount.desc).li
mit(5).show
```

Find the 5 rows with the largest viewCount and a domain of "en", and where the pageName doesn't contain a colon (":")

```
>viewsWithSchemaDF.filter('domain==="en").filter(!'pageName.contains("
:")).orderBy('viewcount.desc).limit(5).show
```

<u>**Summary**</u>

We can see that text data can require a little more work than data like JSON with a pre-existing structure. Once you've restructured it with a clear schema, which is not usually difficult, then the full power of DataFrames can be easily applied

Exercise 9: Dataframe, Datasets and Spark SQL Exploring Grouping

Overview

In this lab, we will work with the Github archive that contains activity for a single day. We'll analyze it to find the top contributors for that day.

Builds on

None - but you must have the spark shell running.

Run time

25-35 minutes

Github Activity Archive

You've already worked with the data file (***spark-labs/data/github.json***) that contains a log of all github activity for one day. In previous labs you should already have

Loaded the data, and viewed the schema.

Selected the actor column, which has a nested structure, and worked with some of its subelements.

We illustrate doing that below.

```
// Load the file into Spark

> val githubDF=spark.read.json("spark-labs/data/github.json")

// Select the actor column

> val actorDF = githubDF.select('actor)

// Print actor schema

> actorDF.printSchema

// Select the actor's login value - note how we

// Use a SQL expression in the select, not a Column

> actorDF.select("actor.login").limit(5).show
```

Assignment: Query the Data by Actor's Login Value

Tasks

- Query the github data for how many entries exist in the data for each actor's login. Use the DSL.

Hints:

  - You'll want to group the data by the actor's login.
  - You'll probably want to use an SQL expression to express the actor's login, not a column value.
  - You want a count of entries for each login value. Show a few rows of this data.

- Lastly, find the 20 logins with the largest number of contributions, and display them.

Assignment 2: Use SQL Tasks

- Optionally, try doing the above query in SQL.
- It's pretty much standard SQL, so if you know that well, it's not very complex.
- Remember to create a temporary view (createOrReplaceTempView)

Summary

The task we did in this lab is not overly complex, but it's also not trivial. Spark SQL makes it reasonable for us to accomplish this in a short lab, either using the DSL, or using SQL.

If you wanted to do this using RDDs, it would be a much more complex series of transformations - starting with the messy ones of parsing the JSON data. This is why Spark SQL is becoming the standard API for working with Spark.

Exercise 10: Seeing Catalyst at Work

Overview
In this lab, we will look at several transformations and examine the optimizations that Catalyst performs.
Builds on
Previous labs for the transformations we'll use.
Run time
20-30 minutes

Lab Preparation
We'll first work with the Wikimedia view data, and see how Catalyst helps to optimize queries involving filtering.
Tasks
You've already worked with the data file (***spark-labs/data/wiki-pageviews.txt***). In previous labs you should already have done the below. If you haven't, then do it now.

- Loaded the data, and then split on whitespace.
- Create a new dataframe with a finer-grained schema.
- We illustrate doing that below.

```
// Load the file into Spark
> val viewsDF=spark.read.text("spark-labs/data/wiki-pageviews.txt")
// Split on whitespace
> val splitViewsDF = viewsDF.select(split('value, "\\s+").as("splitLine"))
// Use a better schema
> val viewsWithSchemaDF = splitViewsDF.select('splitLine(0).as("domain"),
'splitLine(1).as("pageName"), 'splitLine(2).cast("integer").as("viewCount"),
'splitLine(3).cast("long").as("size"))
```

# Python

```
> from pyspark.sql.functions import *
```

```
# Load the file into Spark

> viewsDF=spark.read.text("spark-labs/data/wiki-pageviews.txt")

# Split on whitespace

> splitViewsDF = viewsDF.select(split(viewsDF.value,
"\\s+").alias("splitLine"))

# Use a better schema

> viewsWithSchemaDF =
splitViewsDF.select(splitViewsDF.splitLine[0].alias("domain"),splitViews
DF.splitLine[1].alias("pageName"),splitViewsDF.splitLine[2].cast("integer
").alias("viewCount"), splitViewsDF.splitLine[3].cast("long").alias("size"))
```

Push Down Predicate

Tasks

- ➢ First, write a transformation to order viewsWithSchemaDF by viewCount .
- ➢ explain the above transformation.
    - Note that there is shuffling involved (an Exchange).
    - It needs to shuffle data to sort the rows.
- ➢ Next, filter viewsWithSchemaDF so you only view rows where the domain starts with "en".
    - Put the filter at the end of the transformation chain in your code (after the ordering).
    - You can use the following on the domain column to accomplish this.
        - **Scala:** startsWith("en") (Uppercase W)
        - **Python:** startswith("en") (Lowercase w)
        - explain this transformation.
        - You should see the filtering happening before the shuffle for ordering.
        - Catalyst has **pushed the filter down** to improve efficiency.
        - View the steps that Catalyst took with the detailed version of explain()
            - **Scala:** explain(true)
            - **Python:** explain(True)
- ➢ Optionally, try the transformation with the filter placed before the ordering.
    - It should give you exactly the same plan.

Scala Only: Work with Datasets and lambdas

We'll now create a Dataset and filter it using a lambda. We'll look at how the lambda affects the Catalyst optimizations.

Tasks

- Declare a case class for the Wikimedia data.
- Create a Dataset using the case class.

We show this code below

```
> case class WikiViews(domain:String, pageName:String, viewCount:Integer, size:Long)

> val viewsDS = viewsWithSchemaDF.as[WikiViews]
```

- Next, filter viewsDS so you only view rows where the domain starts with "en".
  - Put the filter at the end of the transformation chain (after the ordering)
  - You have to use a lambda for this.
  - You can use startsWith("en") on the domain value to accomplish this (this is a Scala function on strings).
  - explain this transformation.

```
viewsDS.orderBy('viewCount).filter(view=>view.domain.startsWith("en")).explain
```

- Where does the filtering happen? It should be the very last thing. Why?

Summary Catalyst rules!

We've explained it, now you see it. But lambdas can overthrow the ruler - so be cautious with them.

Exercise 11: Joins and Broadcast

Overview

In this lab, we will explore some slightly more complex queries, including possible alternatives that use a join.

We'll also look at Catalyst's broadcast optimizations when doing joins, and examine performance with those optimizations in place and without them.

Builds on

None

Run time

30-40 minutes

Top 20 Contributors - Straightforward Query

We want to find out the 20 contributors who have the most entries in the github data. Let's first do this in a straightforward way, as done in an earlier lab.

Tasks

- Find the top 20 contributors as follows.

```
// Scala

> val scanQuery =
githubDF.groupBy("actor.login").count.orderBy('count.desc).limit(20)

> scanQuery.show
```

#Python

```
> scanQuery =
githubDF.groupBy("actor.login").count().orderBy("count",ascending=False).limit(
20)

> scanQuery.show()
```

- ➢ Open the Web UI (localhost:4040) and look at the jobs tab.
- ➢ Note the execution time (0.6s on our system)
- ➢ Drill down on this job, and note the shuffle data (333KB on our system)

Top 20 Contributors - Join Query

We've decided that we want to track 20 specific contirbutors (the top 20 contributors from the previous month), instead of the top 20 contributors at any given moment. We keep the login values of these contributors-of-interest in another data file we supply (***spark-labs/data/github-top20.json***)

Write a join query that gives the actor login and count from the entries in *github.json* for the login values in *github-top20.json*.

Tasks(Assignment)

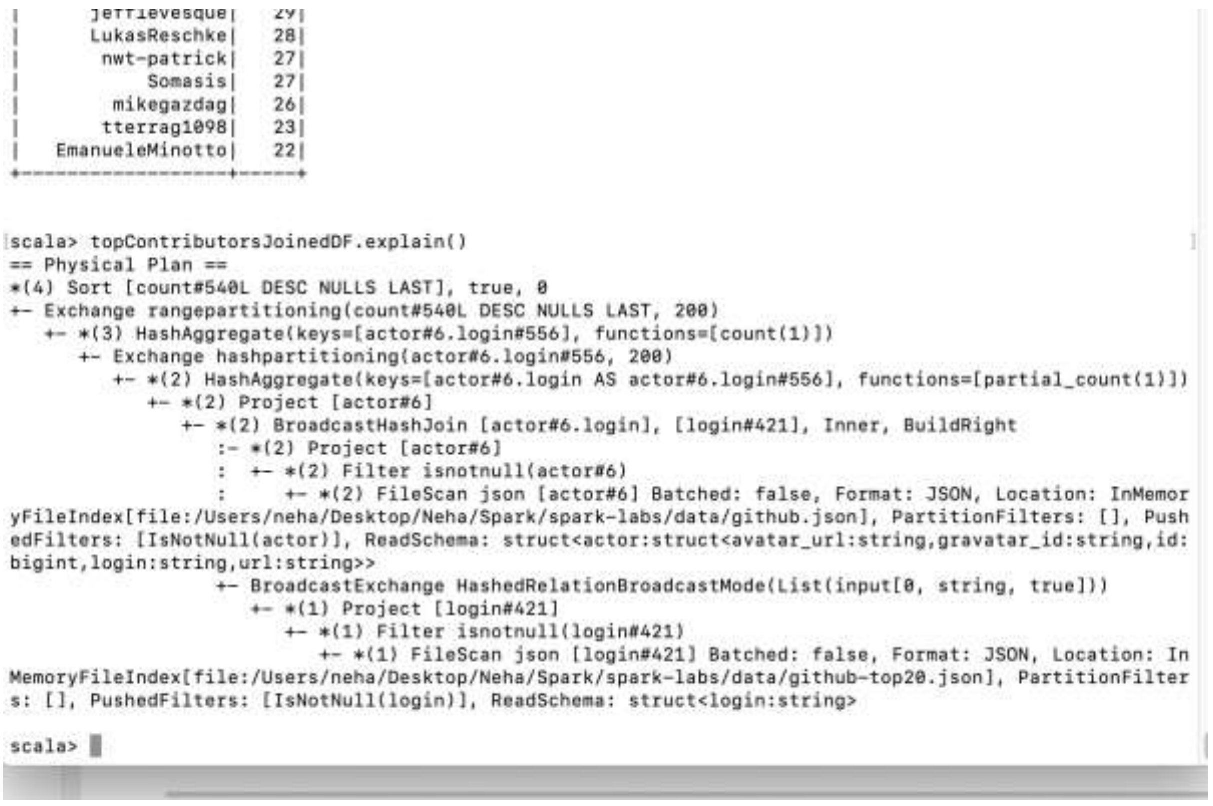
- ➢ Review the data in *github-top20.json*.
- ➢ Accomplish our query needs by:
  - Loading *github-top20.json* into a dataframe (githubTop20DF)
  - Joining githubDF and githubTop20DF in topContributorsJoinedDF
  - Grouping by actor.login
  - Counting the results, and ordering by descending count.

Join Query Performance

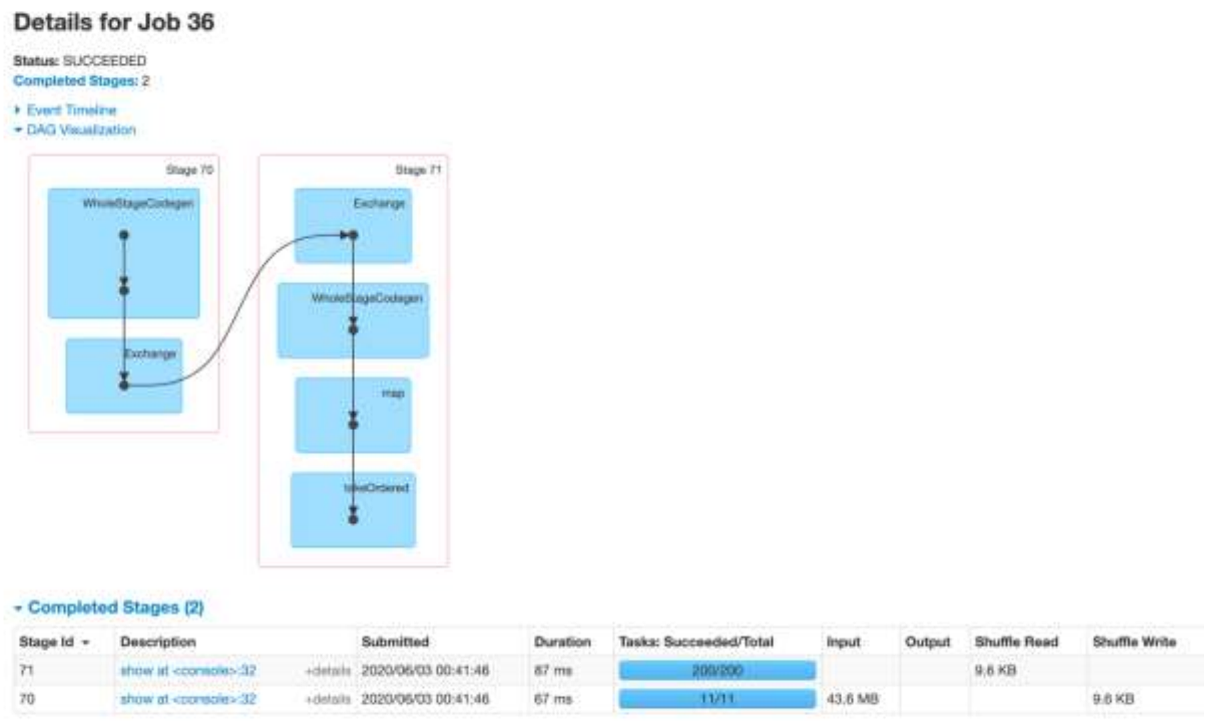
Let's review the performance of this join query, and the optimizations that are being done by Catalyst.

Tasks

- ➢ Open a browser window on the Web UI - localhost:4040.
  - Note the time taken for the last show performed above, drill through on the job for it, and note the shu#e data.
  - On our system it took 0.6s and shuffled 7.9K of data.
  - This is much less data shuffling than the scanQuery done earlier (40 times reduction).
- ➢ Execute topContributorsJoinedDF.explain to see what is going on.
  - We illustrate this below. Note the broadcasts which we have highlighted.
  - Catalyst has optimized the join to use broadcast.



```
|      jeffievesque|    29|
|      LukasReschke|    28|
|       nwt-patrick|    27|
|           Somasis|    27|
|        mikegazdag|    26|
|       tterrag1098|    23|
|   EmanueleMinotto|    22|
+------------------+------+


scala> topContributorsJoinedDF.explain()
== Physical Plan ==
*(4) Sort [count#540L DESC NULLS LAST], true, 0
+- Exchange rangepartitioning(count#540L DESC NULLS LAST, 200)
   +- *(3) HashAggregate(keys=[actor#6.login#556], functions=[count(1)])
      +- Exchange hashpartitioning(actor#6.login#556, 200)
         +- *(2) HashAggregate(keys=[actor#6.login AS actor#6.login#556], functions=[partial_count(1)])
            +- *(2) Project [actor#6]
               +- *(2) BroadcastHashJoin [actor#6.login], [login#421], Inner, BuildRight
                  :- *(2) Project [actor#6]
                  :  +- *(2) Filter isnotnull(actor#6)
                  :     +- *(2) FileScan json [actor#6] Batched: false, Format: JSON, Location: InMemor
yFileIndex[file:/Users/neha/Desktop/Neha/Spark/spark-labs/data/github.json], PartitionFilters: [], Push
edFilters: [IsNotNull(actor)], ReadSchema: struct<actor:struct<avatar_url:string,gravatar_id:string,id:
bigint,login:string,url:string>>
                  +- BroadcastExchange HashedRelationBroadcastMode(List(input[0, string, true]))
                     +- *(1) Project [login#421]
                        +- *(1) Filter isnotnull(login#421)
                           +- *(1) FileScan json [login#421] Batched: false, Format: JSON, Location: In
MemoryFileIndex[file:/Users/neha/Desktop/Neha/Spark/spark-labs/data/github-top20.json], PartitionFilter
s: [], PushedFilters: [IsNotNull(login)], ReadSchema: struct<login:string>

scala> 
```



Details for Job 36

Status: SUCCEEDED
Completed Stages: 2

▸ Event Timeline
▾ DAG Visualization

▾ Completed Stages (2)

Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
71	show at <console>:32	+details	2020/06/03 00:41:46	87 ms	200/200			9.6 KB	
70	show at <console>:32	+details	2020/06/03 00:41:46	67 ms	11/11	43.6 MB			9.6 KB

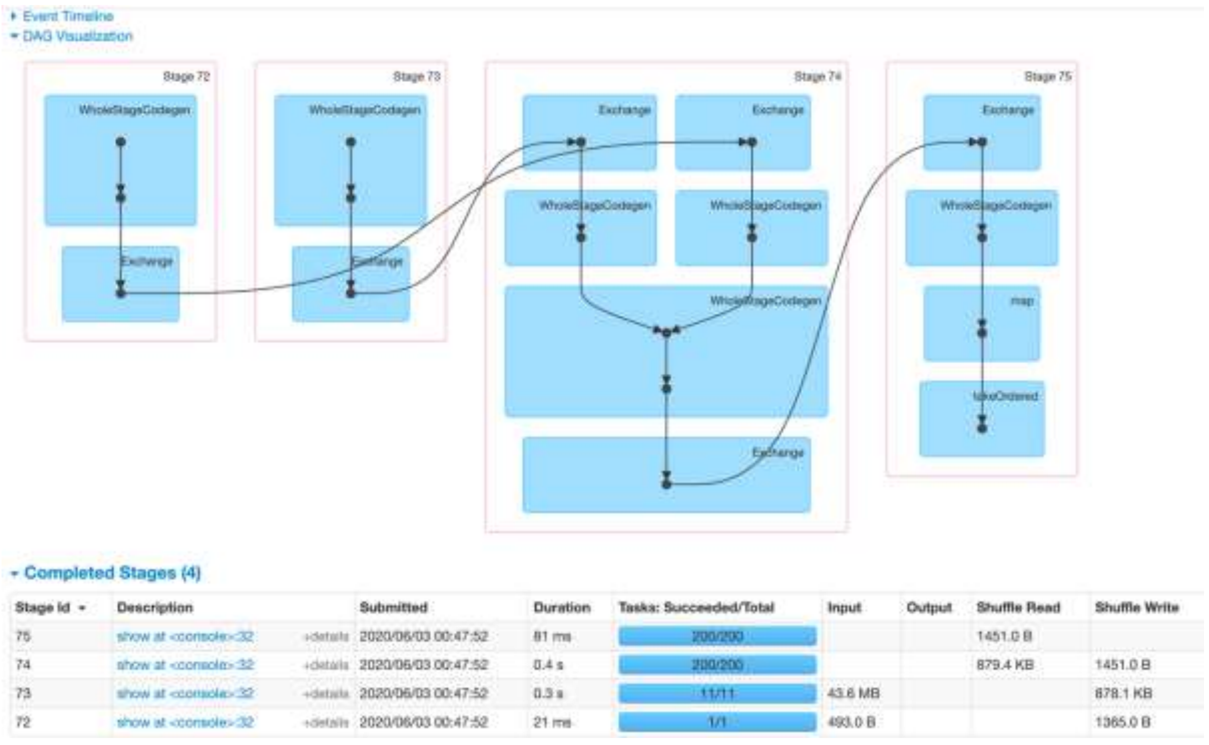
Join Performance without Broadcast

Let's disable the Catalyst broadcast optimization and look at what happens.

Tasks:

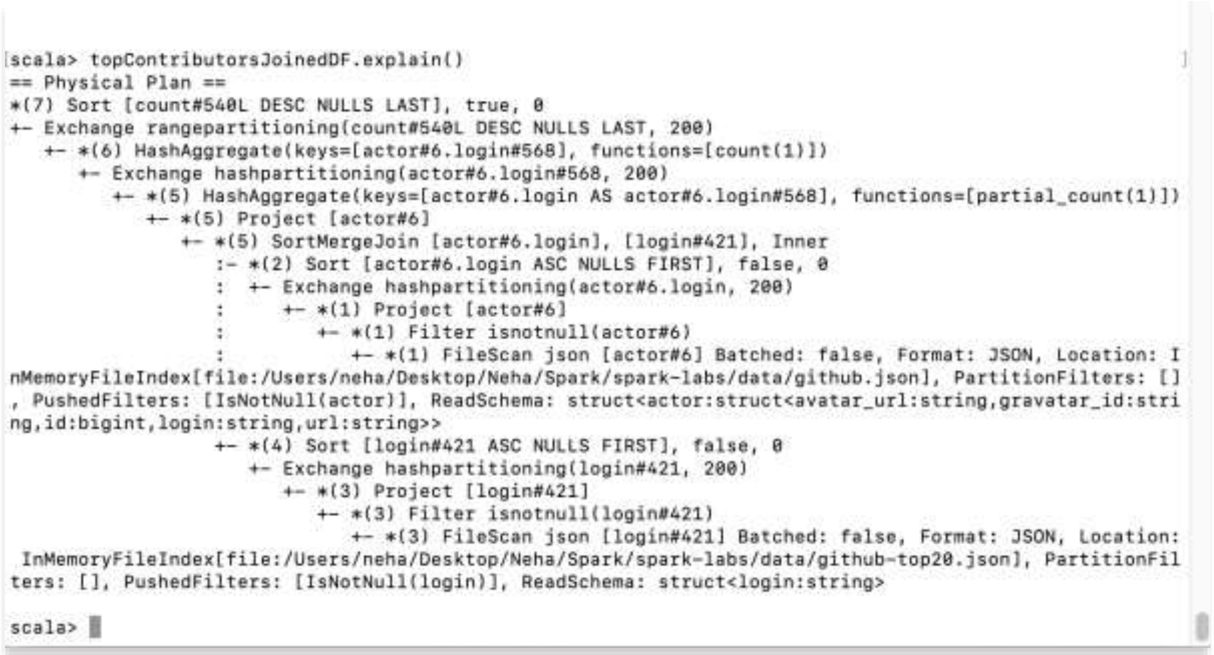
- ➢ Disable automatic broadcasting, by seting the appropriate configuration property as shown below.
- ➢ Next, create and show the topContributorsJoinedDF query again
  - You must create the DataFrame after setting the config property, to make sure it conforms to the new setting.

```
spark.conf.set("spark.sql.autoBroadcastJoinThreshold",-1)
```



▾ Completed Stages (4)

Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
75	show at <console>:32	+details	2020/06/03 00:47:52	81 ms	200/200			1451.0 B	
74	show at <console>:32	+details	2020/06/03 00:47:52	0.4 s	200/200			879.4 KB	1451.0 B
73	show at <console>:32	+details	2020/06/03 00:47:52	0.3 s	11/11	43.6 MB			878.1 KB
72	show at <console>:32	+details	2020/06/03 00:47:52	21 ms	1/1	493.0 B			1365.0 B



```
scala> topContributorsJoinedDF.explain()
== Physical Plan ==
*(7) Sort [count#540L DESC NULLS LAST], true, 0
+- Exchange rangepartitioning(count#540L DESC NULLS LAST, 200)
   +- *(6) HashAggregate(keys=[actor#6.login#568], functions=[count(1)])
      +- Exchange hashpartitioning(actor#6.login#568, 200)
         +- *(5) HashAggregate(keys=[actor#6.login AS actor#6.login#568], functions=[partial_count(1)])
            +- *(5) Project [actor#6]
               +- *(5) SortMergeJoin [actor#6.login], [login#421], Inner
                  :- *(2) Sort [actor#6.login ASC NULLS FIRST], false, 0
                  :  +- Exchange hashpartitioning(actor#6.login, 200)
                  :     +- *(1) Project [actor#6]
                  :        +- *(1) Filter isnotnull(actor#6)
                  :           +- *(1) FileScan json [actor#6] Batched: false, Format: JSON, Location: I
nMemoryFileIndex[file:/Users/neha/Desktop/Neha/Spark/spark-labs/data/github.json], PartitionFilters: []
, PushedFilters: [IsNotNull(actor)], ReadSchema: struct<actor:struct<avatar_url:string,gravatar_id:stri
ng,id:bigint,login:string,url:string>>
                  +- *(4) Sort [login#421 ASC NULLS FIRST], false, 0
                     +- Exchange hashpartitioning(login#421, 200)
                        +- *(3) Project [login#421]
                           +- *(3) Filter isnotnull(login#421)
                              +- *(3) FileScan json [login#421] Batched: false, Format: JSON, Location:
 InMemoryFileIndex[file:/Users/neha/Desktop/Neha/Spark/spark-labs/data/github-top20.json], PartitionFil
ters: [], PushedFilters: [IsNotNull(login)], ReadSchema: struct<login:string>

scala> 
```

- Observe the execution time and shuffled data. This is almost **100 times** the amount of shuffle data!
- Renable automatic broadcasting, by seting the appropriate configuration property as shown below.

```
> spark.conf.set("spark.sql.autoBroadcastJoinThreshold",1024*1024*10)
```

Summary

Joins are powerful tools, but distributed joins can be expensive in terms of amount of data shu#ed. We've worked with them here, and seen how Catalyst can optimize by broadcasting a small data set.

Note that we've used a common strategy here - we've replaced a query that was resource intensive (our straight scan query), with a di!erent query (querying on a specific group of logins that are provided). The new query is not exactly equivalent, but fills our needs, and is much less resource intensive.

Exercise 12: Hive with Spark integration

- Let's create table "reports" in the hive. I am using boa schema in which I am creating a table.

  Enter in to hive CLI and use below commands to create a table:

```
CREATE schema boa;

CREATE TABLE boa.reports(id INT,days INT,YEAR INT);

INSERT INTO TABLE boa.reports VALUES
(121,232,2015),(122,245,2015),(123,134,2014),(126,67,2016),(182,122,2016),(137,9
2,2015),(101,311,2015);

select * from boa.reports;
```

- Go to spark-shell:

  Now, create a data frame hiveReports using below command:

```
var hiveReports = spark.sqlContext.sql("select * from boa.reports")
```

- Check whether dataset report is loaded into data frame hiveReport or not using below command:

  Check schema of Data Frame.

- hiveReports.show() will show the same results as "select * from boa.reports" in Hive CLI.

- Accessing Catalog from spark-shell
  Catalog is available on spark session. The following code shows how to access catalog.

```
val catalog = sparkSession.catalog
```

- Querying the databases
  Once we have access to catalog, we can use it to query the databases. All the API's on catalog returns a dataset.

```
catalog.listDatabases().select("name").show()
```

- Registering Dataframe with createTempView
  In earlier versions of spark, we used to register a dataframe using registerTempTable. But in spark 2.0, this API is deprecated. The registerTempleTable API was one of the source of confusion as users used think it materializes the dataframe and saves as a temporary table which was not the case. So this API is replaced with createTempView.

  createTempView can be used as follows.

```
df.createTempView("sales")

spark.catalog.listTables("boa").show()
```

  Once we have registered a view, we can query it using listTables.

- Checking is table cached or not
  Catalog not only is used for querying. It can be used to check state of individual tables. Given a table, we can check is it cache or not. It's useful in scenarios to make sure we cache the tables which are accessed frequently.

```
catalog.isCached("sales")
```

  You will get false as by default no table will be cache. Now we cache the table and query again.

```
df.cache()

catalog.isCached("sales")
```

  Now it will print true.

- Drop view
  We can use catalog to drop views. In spark sql case, it will deregister the view. In case of hive, it will drops from the metadata store.

```
catalog.dropTempView("sales")
```

Query registered functions
Catalog API not only allow us to interact with tables, it also allows us to interact with udf's. The below code shows how to query all functions registered on spark session. They also include all built in functions.

```
catalog.listFunctions().select("name","description","className","isTemporary").s
how(100)
```

Exercise 13: SQL Tables and Views

CREATING A MANAGED TABLE

Tables reside within a database. By default, Spark creates tables under the default database. To create your own database name, you can issue a SQL command from your Spark application or notebook. Using the US flight delays data set, let's create both a managed and an unmanaged table. To begin, we'll create a database called training_spark_db and tell Spark we want to use that database:

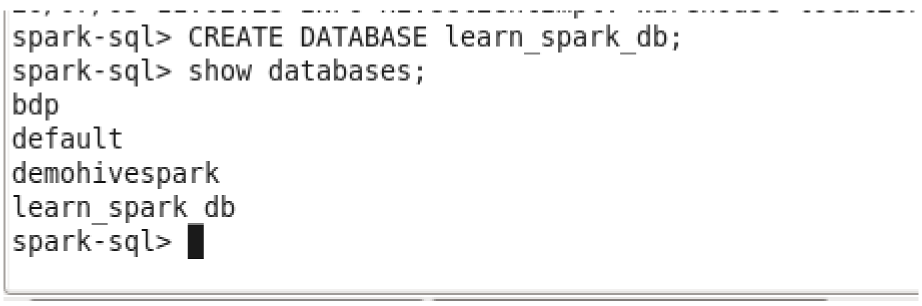
// In Scala/Python

Launch

```
bin/spark-sql -S
```

for spark-sql console

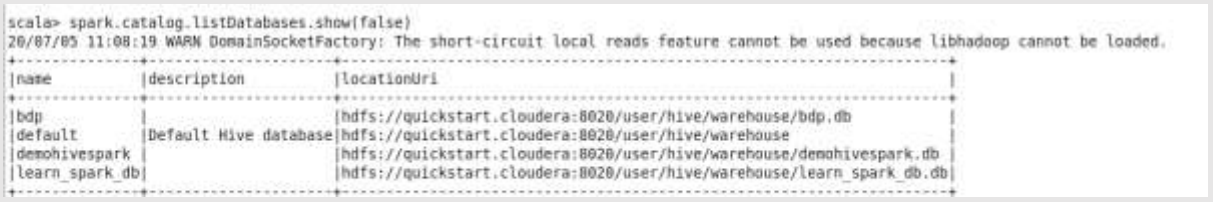
```
CREATE DATABASE learn_spark_db
```



```
spark-sql> CREATE DATABASE learn_spark_db;
spark-sql> show databases;
bdp
default
demohivespark
learn_spark_db
spark-sql> █
```

Launch bin/spark-shell

spark.catalog.listDatabases.show(false)



```
scala> spark.catalog.listDatabases.show(false)
20/07/05 11:08:19 WARN DomainSocketFactory: The short-circuit local reads feature cannot be used because libhadoop cannot be loaded.
+-------------+---------------------+--------------------------------------------------------------+
|name         |description          |locationUri                                                   |
+-------------+---------------------+--------------------------------------------------------------+
|bdp          |                     |hdfs://quickstart.cloudera:8020/user/hive/warehouse/bdp.db    |
|default      |Default Hive database|hdfs://quickstart.cloudera:8020/user/hive/warehouse           |
|demohivespark |                    |hdfs://quickstart.cloudera:8020/user/hive/warehouse/demohivespark.db |
|learn_spark_db|                    |hdfs://quickstart.cloudera:8020/user/hive/warehouse/learn_spark_db.db|
+-------------+---------------------+--------------------------------------------------------------+
```

spark.sql("USE training_spark_db")

spark.sql("CREATE TABLE managed_us_delay_flights_tbl (date STRING, delay INT, distance INT, origin STRING, destination STRING)")

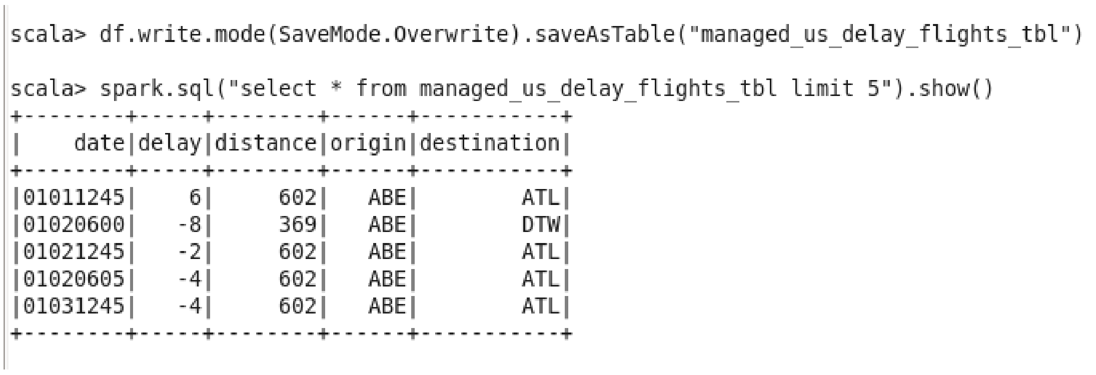
val csvFile="file:///home/cloudera/Desktop/SW/in/Data/departuredelays.csv"

val schema= date STRING, delay INT, distance INT, origin STRING, destination STRING"

val df = spark.read.format("csv").option("schema",schema).option("header", "true").load(csvFile)

import org.apache.spark.sql.{Row, SaveMode }

df.write.mode(SaveMode.Overwrite).saveAsTable("managed_us_delay_flights_tbl ")



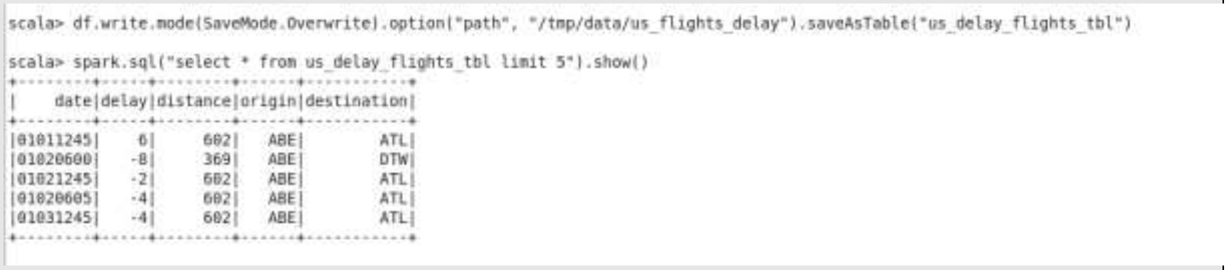
```
scala> df.write.mode(SaveMode.Overwrite).saveAsTable("managed_us_delay_flights_tbl")

scala> spark.sql("select * from managed_us_delay_flights_tbl limit 5").show()
+--------+-----+--------+------+-----------+
|    date|delay|distance|origin|destination|
+--------+-----+--------+------+-----------+
|01011245|    6|     602|   ABE|        ATL|
|01020600|   -8|     369|   ABE|        DTW|
|01021245|   -2|     602|   ABE|        ATL|
|01020605|   -4|     602|   ABE|        ATL|
|01031245|   -4|     602|   ABE|        ATL|
+--------+-----+--------+------+-----------+
```

CREATING AN UNMANAGED TABLE

By contrast, you can create unmanaged tables from your own data sources—say, Parquet, CSV, or JSON files stored in a file store accessible to your Spark application.

To create an unmanaged table from a data source such as a CSV file, in SQL use:

```
spark.sql("""CREATE TABLE us_delay_flights_tbl(date STRING, delay INT,
  distance INT, origin STRING, destination STRING)
  USING csv OPTIONS (PATH
  'file:///home/cloudera/Desktop/SW/in/Data/departuredelays.csv')""")


df.write.mode(SaveMode.Overwrite).option("path",
"/tmp/data/us_flights_delay").saveAsTable("us_delay_flights_tbl")
```



```
scala> df.write.mode(SaveMode.Overwrite).option("path", "/tmp/data/us_flights_delay").saveAsTable("us_delay_flights_tbl")

scala> spark.sql("select * from us_delay_flights_tbl limit 5").show()
+--------+-----+--------+------+-----------+
|    date|delay|distance|origin|destination|
+--------+-----+--------+------+-----------+
|01011245|    6|     602|   ABE|        ATL|
|01020600|   -8|     369|   ABE|        DTW|
|01021245|   -2|     602|   ABE|        ATL|
|01020605|   -4|     602|   ABE|        ATL|
|01031245|   -4|     602|   ABE|        ATL|
+--------+-----+--------+------+-----------+
```

Exercise 14: Hive with Spark integration using IntelliJ

Update pom.xml with the below dependency for hive libraries

```
<dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-hive_${scala.tools.version}</artifactId>

        <version>${spark.version}</version>

        </dependency>
```

Let us create a Scala Object in the existing workspace of training-spark, with the below provided code.

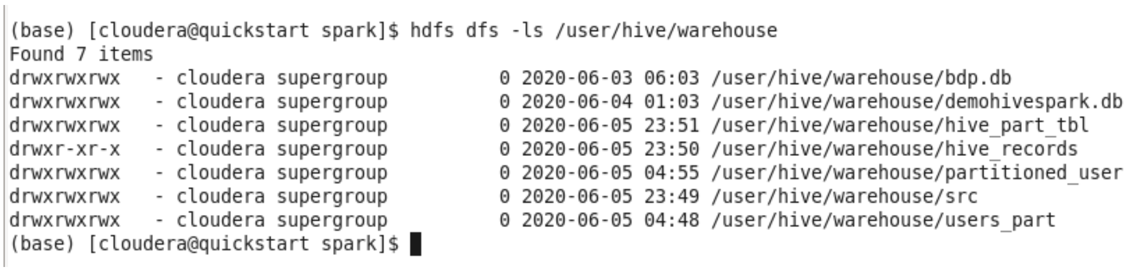
```
import java.io.File


import org.apache.spark.sql.{Row, SaveMode, SparkSession}


case class Record(key: Int, value: String)

// warehouseLocation points to the default location for managed databases and
tables

val warehouseLocation = new File("spark-warehouse").getAbsolutePath


val spark = SparkSession

  .builder()

  .appName("Spark Hive Example") .master("local[*]")

 // .config("spark.sql.warehouse.dir", warehouseLocation)

  .enableHiveSupport()

  .getOrCreate()


import spark.implicits._

import spark.sql


sql("CREATE TABLE IF NOT EXISTS src (key INT, value STRING) USING
hive")

sql("LOAD DATA LOCAL INPATH 'examples/src/main/resources/kv1.txt' INTO
TABLE src")


// Queries are expressed in HiveQL

sql("SELECT * FROM src").show()

// +---+-------+
// |key|  value|
// +---+-------+
// |238|val_238|
// | 86| val_86|
// |311|val_311|
// ...

// Aggregation queries are also supported.
sql("SELECT COUNT(*) FROM src").show()
// +--------+
// |count(1)|
// +--------+
// |    500 |
// +--------+

// The results of SQL queries are themselves DataFrames and support all normal
functions.
val sqlDF = sql("SELECT key, value FROM src WHERE key < 10 ORDER BY
key")

// The items in DataFrames are of type Row, which allows you to access each
column by ordinal.
val stringsDS = sqlDF.map {
  case Row(key: Int, value: String) => s"Key: $key, Value: $value"
}
stringsDS.show()
// +--------------------+
// |               value|
// +--------------------+
// |Key: 0, Value: val_0|
// |Key: 0, Value: val_0|
// |Key: 0, Value: val_0|
// ...


// You can also use DataFrames to create temporary views within a
SparkSession.
val recordsDF = spark.createDataFrame((1 to 100).map(i => Record(i, s"val_$i")))
recordsDF.createOrReplaceTempView("records")


// Queries can then join DataFrame data with data stored in Hive.
sql("SELECT * FROM records r JOIN src s ON r.key = s.key").show()
// +---+------+---+------+
// |key| value|key| value|
// +---+------+---+------+
// |  2| val_2|  2| val_2|
// |  4| val_4|  4| val_4|
// |  5| val_5|  5| val_5|
// ...


// Create a Hive managed Parquet table, with HQL syntax instead of the Spark
SQL native syntax
// `USING hive`
sql("CREATE TABLE hive_records(key int, value string) STORED AS
PARQUET")
// Save DataFrame to the Hive managed table
val df = spark.table("src")
df.write.mode(SaveMode.Overwrite).saveAsTable("hive_records")
// After insertion, the Hive managed table has data now
sql("SELECT * FROM hive_records").show()
// +---+-------+
// |key|  value|
// +---+-------+
// |238|val_238|
// | 86| val_86|
// |311|val_311|
// ...

// Prepare a Parquet data directory
val dataDir = "/tmp/parquet_data"
spark.range(10).write.parquet(dataDir)
// Create a Hive external Parquet table
sql(s"CREATE EXTERNAL TABLE hive_bigints(id bigint) STORED AS
PARQUET LOCATION '$dataDir'")
// The Hive external table should already have data
sql("SELECT * FROM hive_bigints").show()
// +---+
// | id|
// +---+
// |  0|
// |  1|
// |  2|
// ... Order may vary, as spark processes the partitions in parallel.

// Turn on flag for Hive Dynamic Partitioning
spark.sqlContext.setConf("hive.exec.dynamic.partition", "true")
spark.sqlContext.setConf("hive.exec.dynamic.partition.mode", "nonstrict")
// Create a Hive partitioned table using DataFrame API
df.write.partitionBy("key").format("hive").saveAsTable("hive_part_tbl")
// Partitioned column `key` will be moved to the end of the schema.
sql("SELECT * FROM hive_part_tbl").show()
// +-------+---+
// |  value|key|
// +-------+---+
// |val_238|238|
// | val_86| 86|
// |val_311|311|

spark.stop()
```

Clean and build the jar using maven tools and submit t he spark-submit command

```
bin/spark-submit --class
com.sparkTutorial.hiveintegration.HiveIntegrationSpark
/home/cloudera/Desktop/SW/Code/Lesson4/Lesson4/target/trainingSpark-1.0.jar -
-master yarn
```

Observe the output from the console and hive table created from CLI.

The metastore in Hive will store, as below



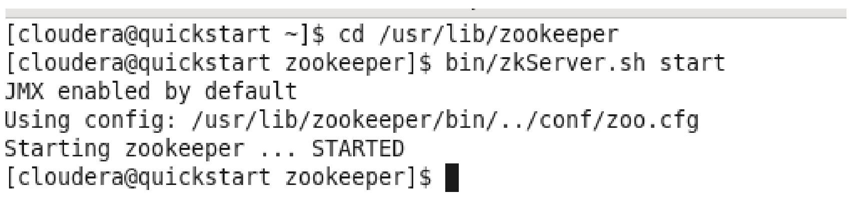
```
(base) [cloudera@quickstart spark]$ hdfs dfs -ls /user/hive/warehouse
Found 7 items
drwxrwxrwx   - cloudera supergroup          0 2020-06-03 06:03 /user/hive/warehouse/bdp.db
drwxrwxrwx   - cloudera supergroup          0 2020-06-04 01:03 /user/hive/warehouse/demohivespark.db
drwxrwxrwx   - cloudera supergroup          0 2020-06-05 23:51 /user/hive/warehouse/hive_part_tbl
drwxr-xr-x   - cloudera supergroup          0 2020-06-05 23:50 /user/hive/warehouse/hive_records
drwxrwxrwx   - cloudera supergroup          0 2020-06-05 04:55 /user/hive/warehouse/partitioned_user
drwxrwxrwx   - cloudera supergroup          0 2020-06-05 23:49 /user/hive/warehouse/src
drwxrwxrwx   - cloudera supergroup          0 2020-06-05 04:48 /user/hive/warehouse/users_part
(base) [cloudera@quickstart spark]$ 
```

Exercise 15: 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
   Navigate to /home/cloudera/Desktop/SW/zookeeper-3.4.14.

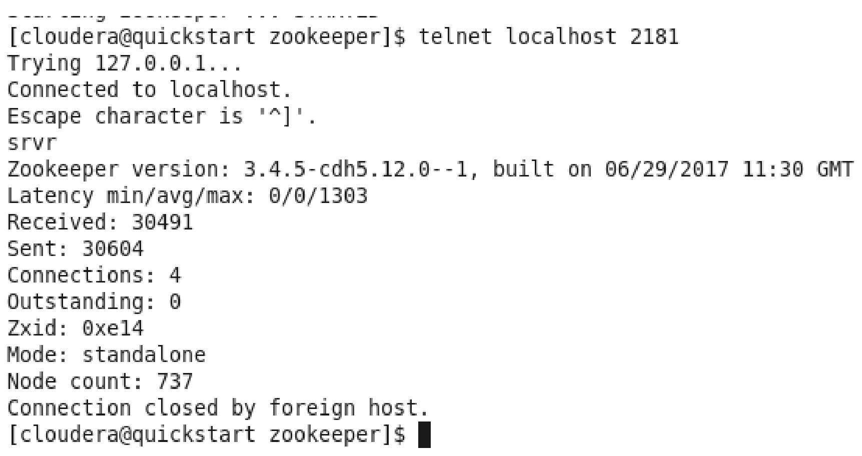
```
bin/zkServer.sh start
```



```
[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
```

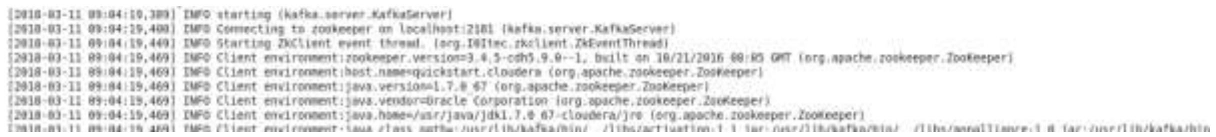


```
[cloudera@quickstart zookeeper]$ telnet localhost 2181
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
srvr
Zookeeper version: 3.4.5-cdh5.12.0--1, built on 06/29/2017 11:30 GMT
Latency min/avg/max: 0/0/1303
Received: 30491
Sent: 30604
Connections: 4
Outstanding: 0
Zxid: 0xe14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

Starting Kafka and Creating a Kafka Topic

1. Open a new terminal and start Kafka broker, Change the user to cloudera with su - - and provide password as cloudera.
   Navigate to /home/cloudera/Desktop/SW/kafka_2.11-2.2.1

```
bin/kafka-server-start.sh config/server.properties
```



```
[2018-03-11 09:04:19,389] INFO starting (kafka.server.KafkaServer)
[2018-03-11 09:04:19,400] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2018-03-11 09:04:19,449] INFO Starting ZkClient event thread. (org.I0Itec.zkclient.ZkEventThread)
[2018-03-11 09:04:19,469] INFO Client environment:zookeeper.version=3.4.5-cdh5.9.0--1, built on 10/21/2016 08:05 GMT (org.apache.zookeeper.ZooKeeper)
[2018-03-11 09:04:19,469] INFO Client environment:host.name=quickstart.cloudera (org.apache.zookeeper.ZooKeeper)
[2018-03-11 09:04:19,469] INFO Client environment:java.version=1.7.0_67 (org.apache.zookeeper.ZooKeeper)
[2018-03-11 09:04:19,469] INFO Client environment:java.vendor=Oracle Corporation (org.apache.zookeeper.ZooKeeper)
[2018-03-11 09:04:19,469] INFO Client environment:java.home=/usr/java/jdk1.7.0_67-cloudera/jre (org.apache.zookeeper.ZooKeeper)
[2018-03-11 09:04:19,469] INFO Client environment:java.class.path=:/usr/lib/kafka/bin/../libs/activation-1.1.jar:/usr/lib/kafka/bin/../libs/aopalliance-1.0.jar:/usr/lib/kafka/bin
```

Kafka broker is running on port 9042.

2. 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.sh --create \ --zookeeper localhost:2181 \ --
replication-factor 1 \ --partitions 1 \
--topic weblogs
```

Here, Topic name

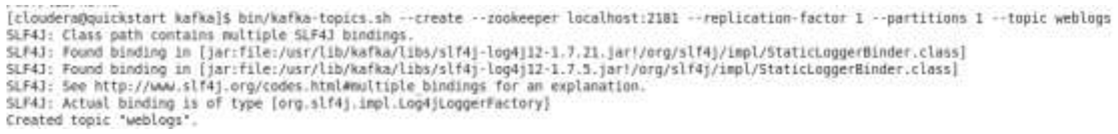
The name of the topic that you wish to create.

Replication Factor

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

Partitions

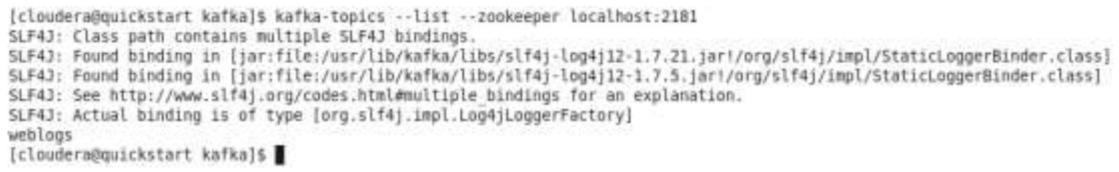
The number of partitions to create for the topic.



```
[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:

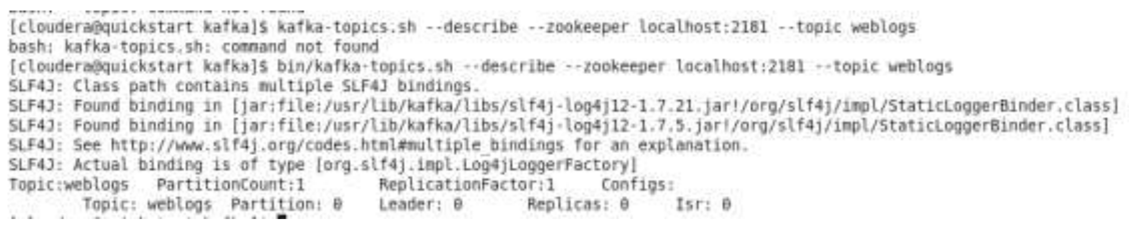
```
$ kafka-topics --list --zookeeper localhost:2181
```



```
[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:

```
$ bin/kafka-console-producer.sh --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
```

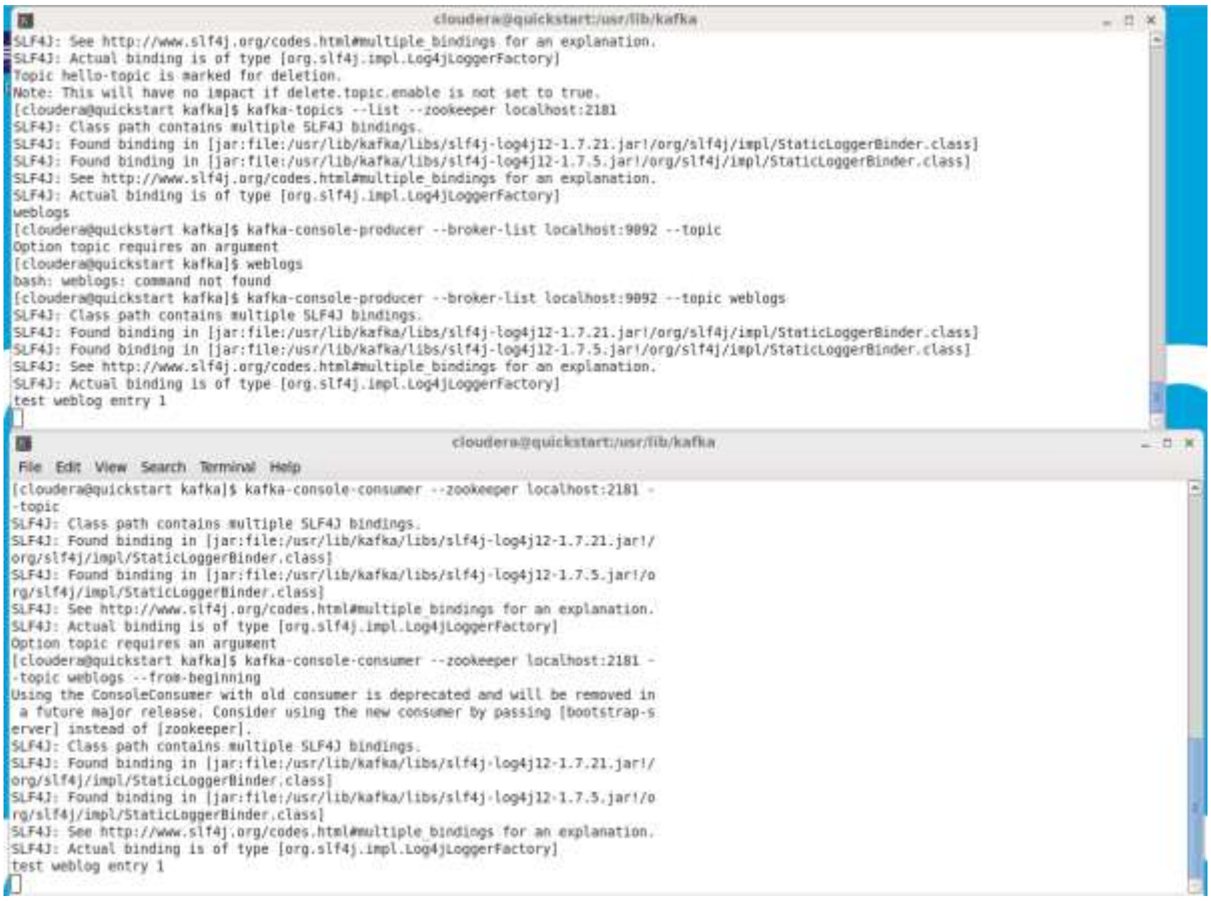
5. Open a new terminal window and adjust it to fit on the window beneath the producer window. Set the title for this window to "Kafka Consumer."

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

```
$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --
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
```



```
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.
[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]$ kafka-console-producer --broker-list localhost:9092 --topic
Option topic requires an argument
[cloudera@quickstart kafka]$ weblogs
bash: weblogs: command not found
[cloudera@quickstart kafka]$ kafka-console-producer --broker-list localhost:9092 --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]
test weblog entry 1
```

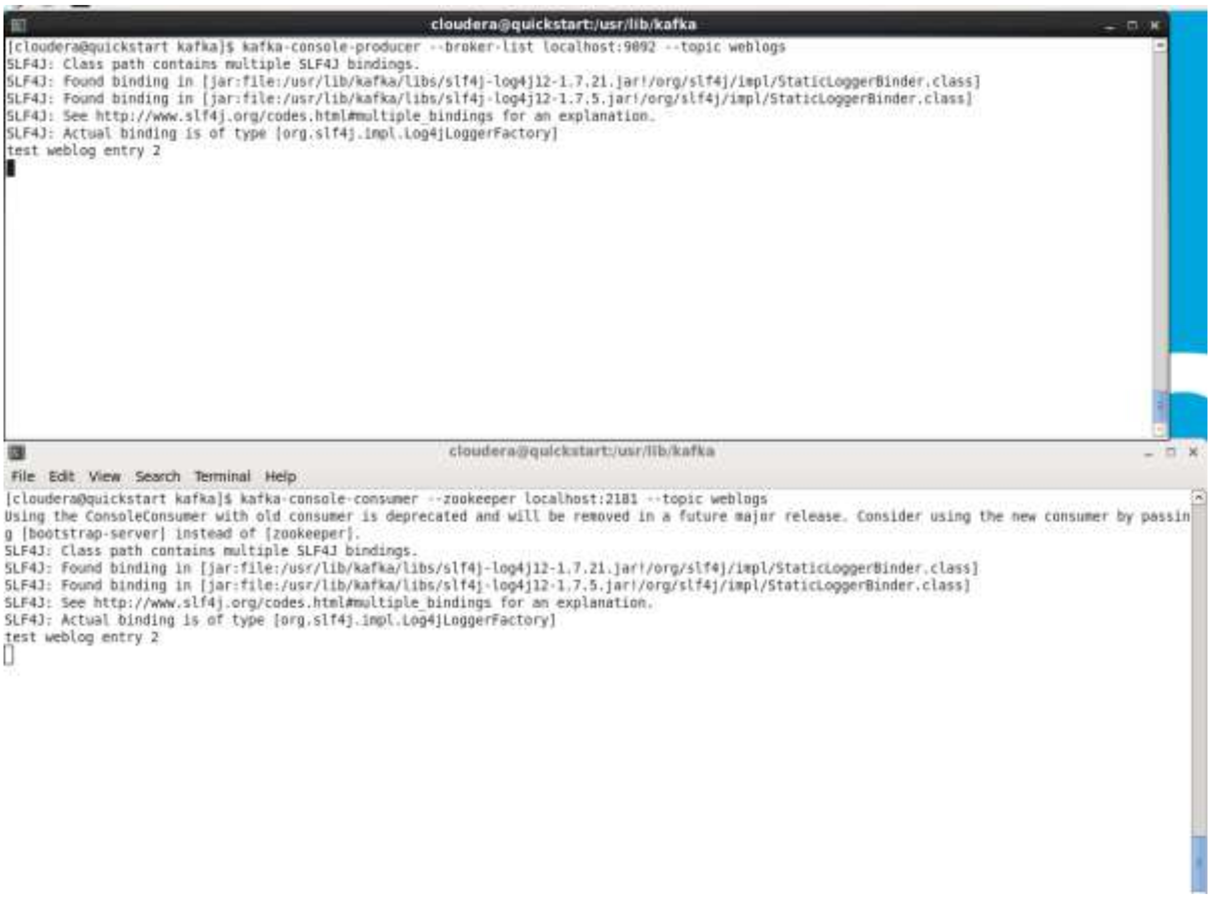
cloudera@quickstart:/usr/lib/kafka
File Edit View Search Terminal Help

```
[cloudera@quickstart kafka]$ kafka-console-consumer --zookeeper localhost:2181 -
-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!/o
rg/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]
Option topic requires an argument
[cloudera@quickstart kafka]$ kafka-console-consumer --zookeeper localhost:2181 -
-topic weblogs --from-beginning
Using the ConsoleConsumer with old consumer is deprecated and will be removed in
 a future major release. Consider using the new consumer by passing [bootstrap-s
erver] instead of [zookeeper].
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!/o
rg/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]
test weblog entry 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
```



```
[cloudera@quickstart kafka]$ kafka-console-producer --broker-list localhost:9092 --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]
test weblog entry 2
```

cloudera@quickstart:/usr/lib/kafka
File Edit View Search Terminal Help

```
[cloudera@quickstart kafka]$ kafka-console-consumer --zookeeper localhost:2181 --topic weblogs
Using the ConsoleConsumer with old consumer is deprecated and will be removed in a future major release. Consider using the new consumer by passin
g [bootstrap-server] instead of [zookeeper].
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]
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.

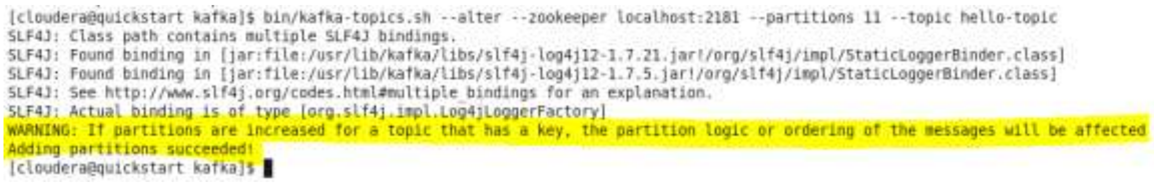
Exercise 16: Alter Apache Kafka Topics

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.

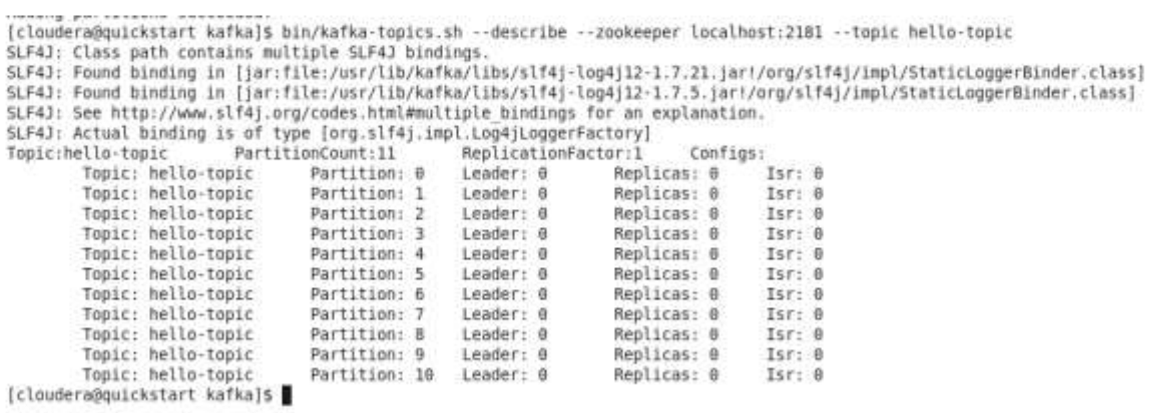
```
bin/kafka-topics.sh --alter --zookeeper localhost:2181 --
partitions 11 --topic hello-topic
```



```
[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

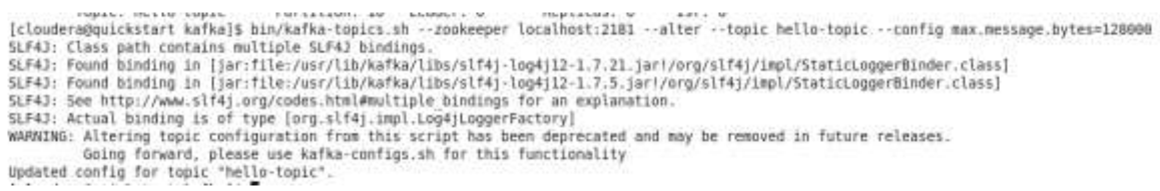
   The general syntax is:

```
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:

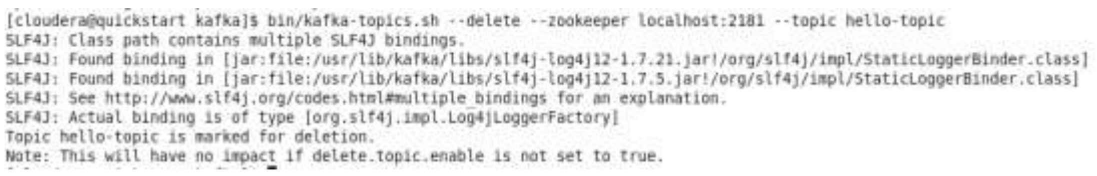
```
bin/kafka-topics.sh --zookeeper localhost:2181 --alter --topic
hello-topic --delete-config max.message.bytes
```

Exercise 18: 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.

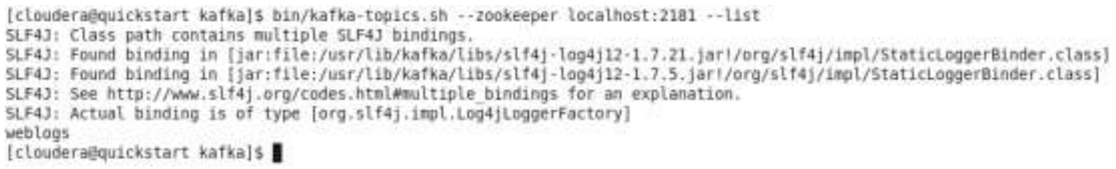
We will delete the topic created with the below command

```
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]$ █
```

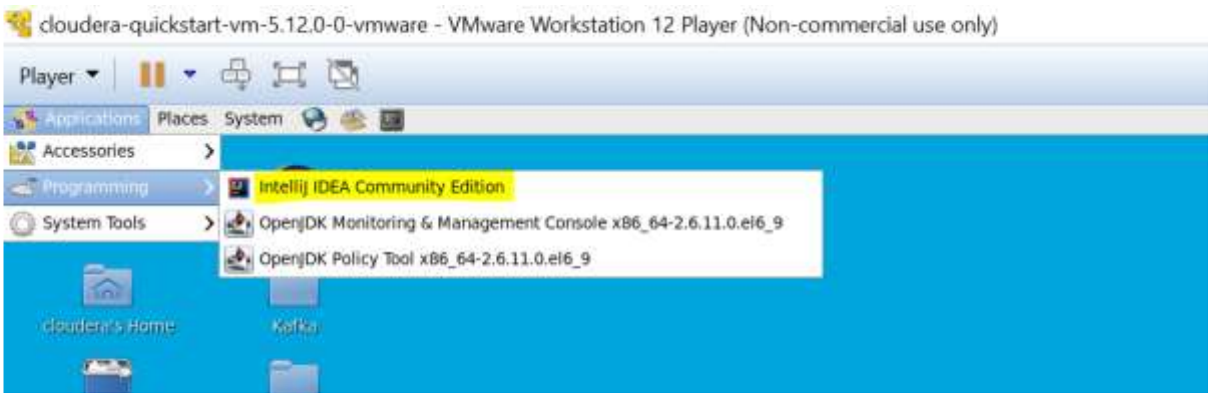
Tip: DATA LOSS AHEAD

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

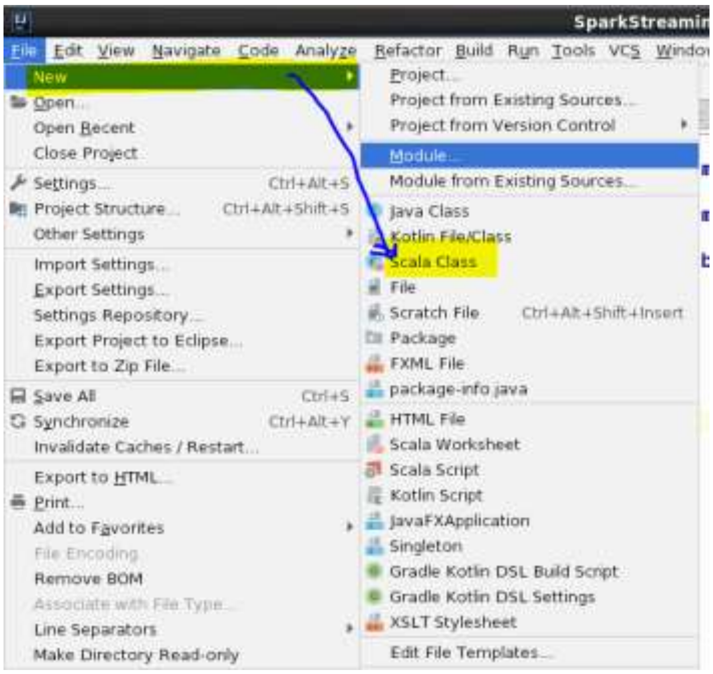
Exercise 19: Building Spark Application using IntelliJ IDE Apache Kafka Scala Client API

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

1. Create new SBT Project using the menu bar option File –>New Project -> Select Scala and SBT. Give a name of your choice for the project.


cloudera-quickstart-vm-5.12.0-0-vmware - VMware Workstation 12 Player (Non-commercial use only)
Player
Applications
Places
System
Accessories
Programming
System Tools
IntelliJ IDEA Community Edition
OpenJDK Monitoring & Management Console x86_64-2.6.11.0.el6_9
OpenJDK Policy Tool x86_64-2.6.11.0.el6_9
cloudera's Home
Kafka

Under src→main→scala, Add a new Scala class.



2. The content of build.sbt

```
version := "1.0"

scalaVersion := "2.11.6"

libraryDependencies ++= Seq (

"org.apache.kafka" %"kafka-clients" %"0.11.0.0

)
```

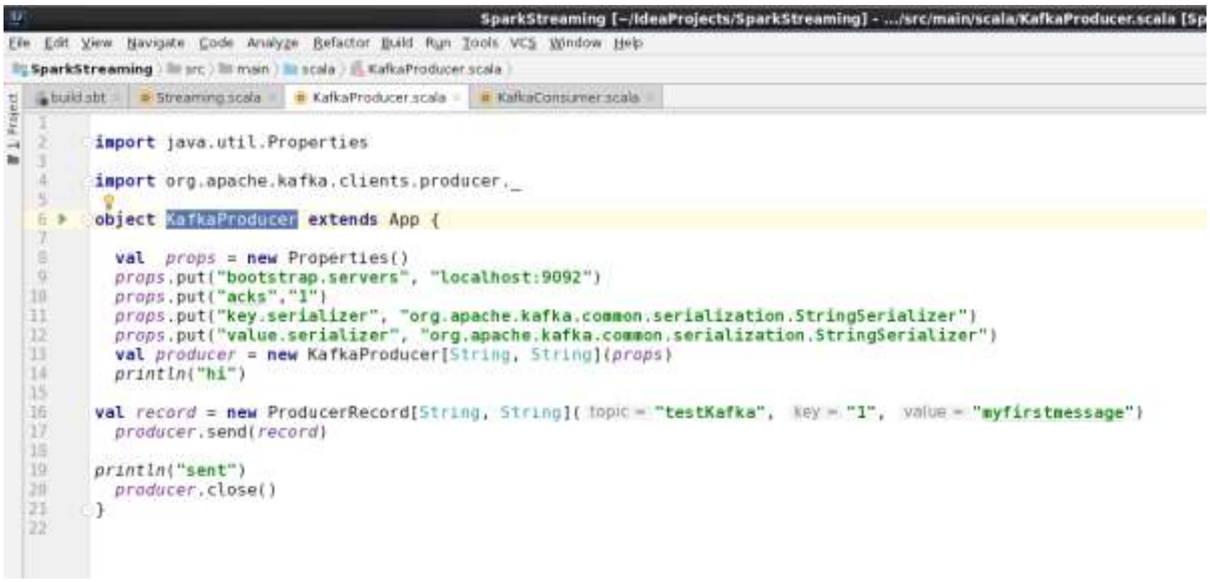
Note: A pop up of “Enable Auto Import”, please click on it to enable it.

3. Create a KafkaProducer Scala Object, as below:-

4. Now add a new Scala class file to project, Right click on project > New > ScalaClass > KafkaProducerTest.

```
object KafkaProducerTest extends App {


   import Scala.util.Properties


    import org.apache.kafka.clients.producer._
```

```
        val  props = new Properties()
          props.put("bootstrap.servers", "localhost:9092")
            props.put("acks","1")
              props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer")
                props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer")


                  val producer = new KafkaProducer[String, String](props)



                    val topic="testKafka"
                      for(i<- 1 to 50) {
                              val record = new ProducerRecord(topic, "key"+i, "value"+i)
                                  producer.send(record)
                                    }
                                println("sent")
                                      producer.close()
}
```


SparkStreaming [~/IdeaProjects/SparkStreaming] - .../src/main/scala/KafkaProducer.scala [Sp
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
SparkStreaming src main scala KafkaProducer.scala
build.sbt Streaming.scala KafkaProducer.scala KafkaConsumer.scala
1: Project

```

import java.util.Properties

import org.apache.kafka.clients.producer._

object KafkaProducer extends App {

  val  props = new Properties()
  props.put("bootstrap.servers", "localhost:9092")
  props.put("acks","1")
  props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer")
  props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer")
  val producer = new KafkaProducer[String, String](props)
  println("hi")

val record = new ProducerRecord[String, String]( topic = "testKafka",  key = "1",  value = "myfirstmessage")
  producer.send(record)

println("sent")
  producer.close()
}

```

<u>A Kafka producer has three mandatory properties</u>:

<u>bootstrap.servers</u>

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.

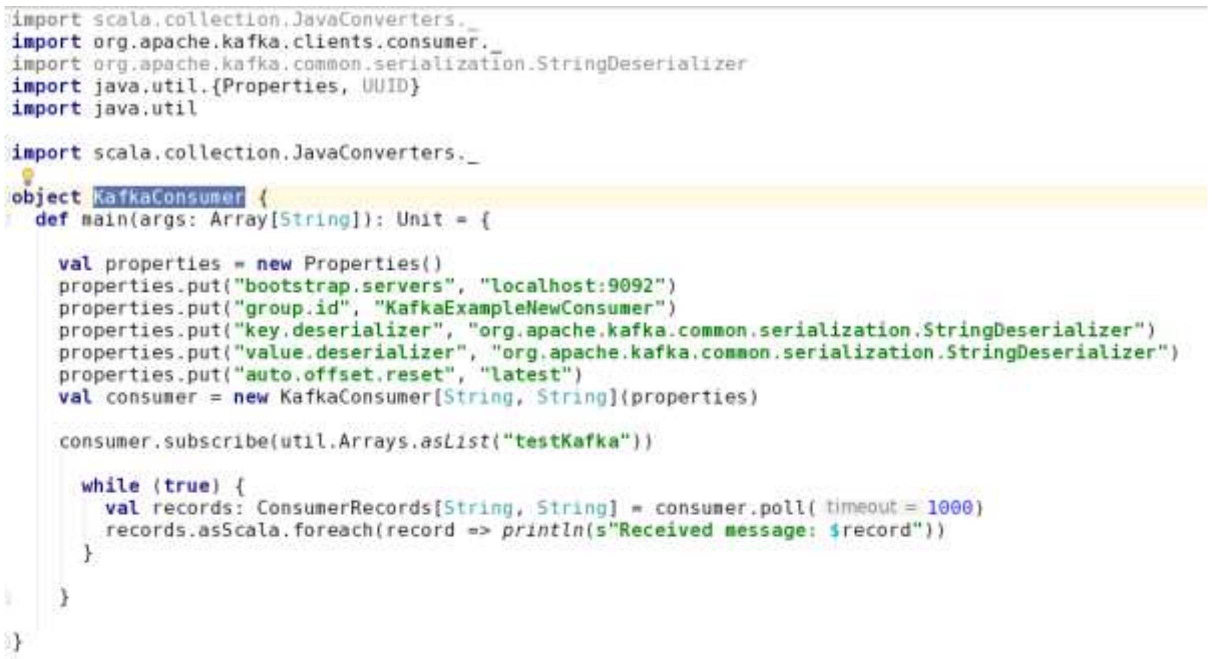
Setting key.serializer is required even if you intend to send only values.

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.

5. Now let us add a new Scala class file to project, Right click on project >New > Class > KafkaConsumerTest

```
import scala.collection.ScalaConverters._

import Scala.util

import Scala.util.Properties


import org.apache.kafka.clients.consumer.{ConsumerRecords, KafkaConsumer}


object KafkaConsumerTest extends App {
  val properties = new Properties()

  properties.put("bootstrap.servers", "localhost:9092")

  properties.put("group.id", "KafkaExampleNewConsumer")

  properties.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")

  properties.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")

  properties.put("auto.offset.reset", "latest")

  val consumer = new KafkaConsumer[String, String](properties)


  consumer.subscribe(util.Arrays.asList("testKafka"))


  while (true) {

    val records: ConsumerRecords[String, String] = consumer.poll(1000)

    records.asScala.foreach(record => println(s"Received message: "+record))

  }


}
```



```
import scala.collection.JavaConverters._
import org.apache.kafka.clients.consumer._
import org.apache.kafka.common.serialization.StringDeserializer
import java.util.{Properties, UUID}
import java.util

import scala.collection.JavaConverters._

object KafkaConsumer {
  def main(args: Array[String]): Unit = {

    val properties = new Properties()
    properties.put("bootstrap.servers", "localhost:9092")
    properties.put("group.id", "KafkaExampleNewConsumer")
    properties.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")
    properties.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")
    properties.put("auto.offset.reset", "latest")
    val consumer = new KafkaConsumer[String, String](properties)

    consumer.subscribe(util.Arrays.asList("testKafka"))

      while (true) {
        val records: ConsumerRecords[String, String] = consumer.poll( timeout = 1000)
        records.asScala.foreach(record => println(s"Received message: $record"))
      }

    }

}
```

6. Now we will test our KafkaProducerTest sending some messages, which will be received by KafkaConsumerTest, before testing ensure that

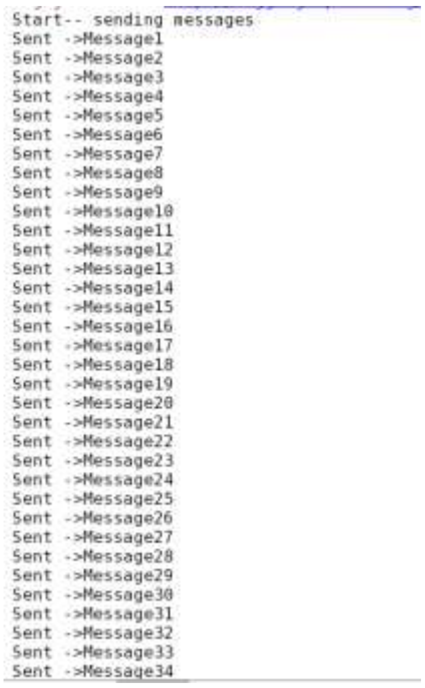
Pre-requisites:-

Topic ‘testKafka’ is created.

Zookeeper should be running on localhost:2181.

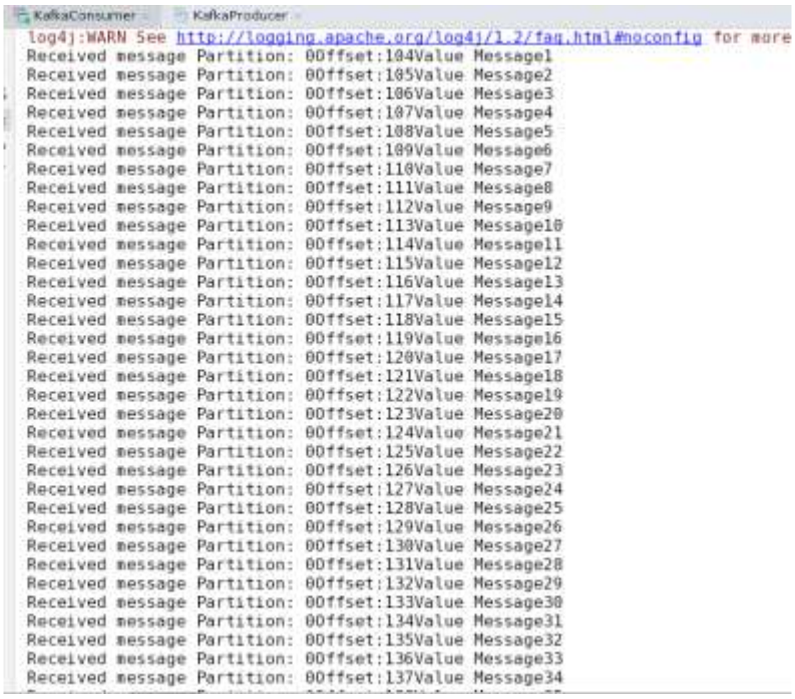
Kafka should be running on localhost:9092.

Run ProducerTest.Scala > Scala Application.



```
Start-- sending messages
Sent ->Message1
Sent ->Message2
Sent ->Message3
Sent ->Message4
Sent ->Message5
Sent ->Message6
Sent ->Message7
Sent ->Message8
Sent ->Message9
Sent ->Message10
Sent ->Message11
Sent ->Message12
Sent ->Message13
Sent ->Message14
Sent ->Message15
Sent ->Message16
Sent ->Message17
Sent ->Message18
Sent ->Message19
Sent ->Message20
Sent ->Message21
Sent ->Message22
Sent ->Message23
Sent ->Message24
Sent ->Message25
Sent ->Message26
Sent ->Message27
Sent ->Message28
Sent ->Message29
Sent ->Message30
Sent ->Message31
Sent ->Message32
Sent ->Message33
Sent ->Message34
```

Right click on the scala class→Run KafkaConsumerTest.scala, the console of the receiver will receive all the messages from the broker. The console will print the key and the message.



```
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more
Received message Partition: 0Offset:104Value Message1
Received message Partition: 0Offset:105Value Message2
Received message Partition: 0Offset:106Value Message3
Received message Partition: 0Offset:107Value Message4
Received message Partition: 0Offset:108Value Message5
Received message Partition: 0Offset:109Value Message6
Received message Partition: 0Offset:110Value Message7
Received message Partition: 0Offset:111Value Message8
Received message Partition: 0Offset:112Value Message9
Received message Partition: 0Offset:113Value Message10
Received message Partition: 0Offset:114Value Message11
Received message Partition: 0Offset:115Value Message12
Received message Partition: 0Offset:116Value Message13
Received message Partition: 0Offset:117Value Message14
Received message Partition: 0Offset:118Value Message15
Received message Partition: 0Offset:119Value Message16
Received message Partition: 0Offset:120Value Message17
Received message Partition: 0Offset:121Value Message18
Received message Partition: 0Offset:122Value Message19
Received message Partition: 0Offset:123Value Message20
Received message Partition: 0Offset:124Value Message21
Received message Partition: 0Offset:125Value Message22
Received message Partition: 0Offset:126Value Message23
Received message Partition: 0Offset:127Value Message24
Received message Partition: 0Offset:128Value Message25
Received message Partition: 0Offset:129Value Message26
Received message Partition: 0Offset:130Value Message27
Received message Partition: 0Offset:131Value Message28
Received message Partition: 0Offset:132Value Message29
Received message Partition: 0Offset:133Value Message30
Received message Partition: 0Offset:134Value Message31
Received message Partition: 0Offset:135Value Message32
Received message Partition: 0Offset:136Value Message33
Received message Partition: 0Offset:137Value Message34
```

Terminate the KafkaConsumerTest.Scala class.

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

1. Create a Producer class as “KafkaPublisher” for the topic “topic-1”.
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

val record: ProducerRecord[String, String] = new ProducerRecord(TOPIC, inlineMessage)

The ProducerRecord objects we created included a topic name and value.

7. Send the message i.e. record using
   val futureResponse: Future[RecordMetadata] = producer.send(record)

   futureResponse.isDone

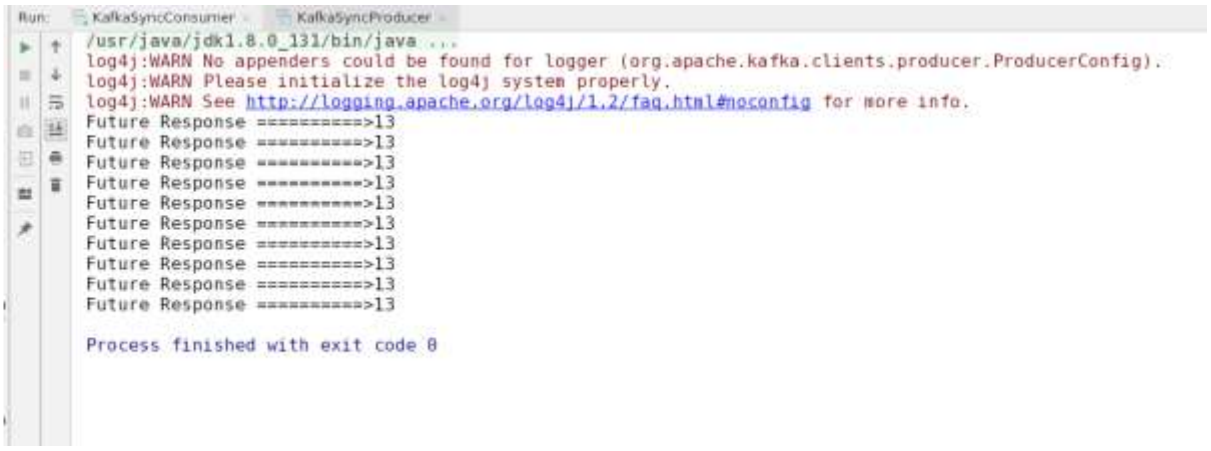
   in the KafkaPublisher.Scala class.
8. Create a Consumer class as KafkaSubscriber.
9. Set the three mandatory properties: bootstrap.servers, key.deserializer, and value.deserializer (deserializer should be of same type as of serializer).
10. Now set the fourth property, which is not strictly mandatory, 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

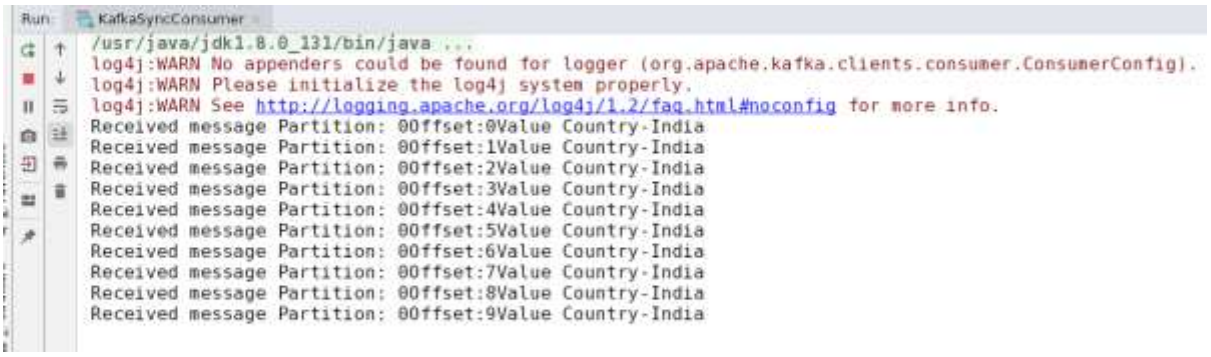
12. Run the KafkaPublisher as Scala Application and also run the KafkaSubscriber as Scala Application.
13. The received message will be printed in the console.

Producer



```
/usr/java/jdk1.8.0_131/bin/java ...
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.producer.ProducerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Future Response ==========>13
Future Response ==========>13
Future Response ==========>13
Future Response ==========>13
Future Response ==========>13
Future Response ==========>13
Future Response ==========>13
Future Response ==========>13
Future Response ==========>13
Future Response ==========>13

Process finished with exit code 0
```

Consumer



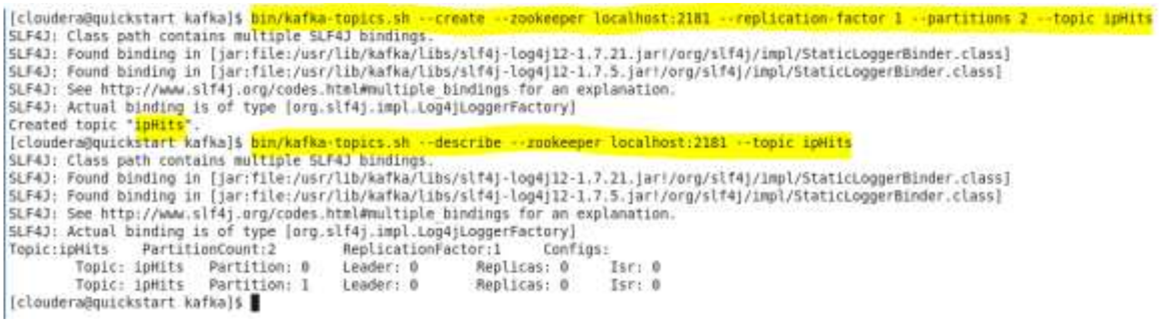
```
/usr/java/jdk1.8.0_131/bin/java ...
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.consumer.ConsumerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Received message Partition: 0Offset:0Value Country-India
Received message Partition: 0Offset:1Value Country-India
Received message Partition: 0Offset:2Value Country-India
Received message Partition: 0Offset:3Value Country-India
Received message Partition: 0Offset:4Value Country-India
Received message Partition: 0Offset:5Value Country-India
Received message Partition: 0Offset:6Value Country-India
Received message Partition: 0Offset:7Value Country-India
Received message Partition: 0Offset:8Value Country-India
Received message Partition: 0Offset:9Value Country-India
```

Exercise 20: 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: Let's jump to the next level by writing another program that implements customized message partitioning. The example consists of recollecting the IPs visiting a web site, which are recorded and published. The message has three parts: web site name, and IP address.

Let us create a topic with two partitions.



```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 2 --topic ipHits
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 "ipHits".
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic ipHits
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:ipHits    PartitionCount:2        ReplicationFactor:1     Configs:
        Topic: ipHits   Partition: 0    Leader: 0       Replicas: 0     Isr: 0
        Topic: ipHits   Partition: 1    Leader: 0       Replicas: 0     Isr: 0
[cloudera@quickstart kafka]$
```

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

1. Create a SBT project and define all the dependencies.
2. Create a Scala class as CountryPartitioner

```
import kafka.producer._


import Scala.util


import org.apache.kafka.clients.producer.KafkaProducer

import org.apache.kafka.clients.producer.Partitioner

import org.apache.kafka.common.Cluster
```

```
object CountryPartitioner  {

  private var producer: KafkaProducer[String, String] = _
}

class CountryPartitioner  extends Partitioner {

  def partition(key: AnyRef, a_numPartitions: Int): Int = {
    var partition = 1
    val partitionKey = key.asInstanceOf[String]
    val offset = partitionKey

    if (offset == "USA") {
      partition = 0
      println("its"+offset +"and its partition is "+partition)
    } else if (offset == "INDIA") {
      partition = 1
      println("its"+offset +"and its partition is "+partition)
    }
    partition
  }

  override def partition(topic: String,
                       key: AnyRef,
                       keyBytes: Array[Byte],
                       value: AnyRef,
                       valueBytes: Array[Byte],
                       cluster: Cluster): Int = partition(key, 10)
  override def close() {

  }


  override def configure(configs: util.Map[String, _]) {

  }

}
```

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 Scala.util.Properties

import org.apache.kafka.clients.producer._


object KafkaProducerwithPartitioner extends App {

  val  props = new Properties()

  props.put("bootstrap.servers", "localhost:9092")

  props.put("acks","1")

  props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer")

  props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer")


  props.put("partitioner.class","CountryPartitioner")


  val producer = new KafkaProducer[String, String](props)

  println("Start-- sending messages")

  var ipCountry="INDIA"
  for(i <- 1 to 100) {
    if( i %2 == 0){
      ipCountry = "USA"


    }
    else{
      ipCountry = "INDIA"
    }


    val record = new ProducerRecord[String, String]("ipHits", ipCountry,"Message" +
ipCountry)
    producer.send(record)
    println("Sent ->Message" + i)
  }


  println("Sent--closing connection")
  producer.close()
}
```

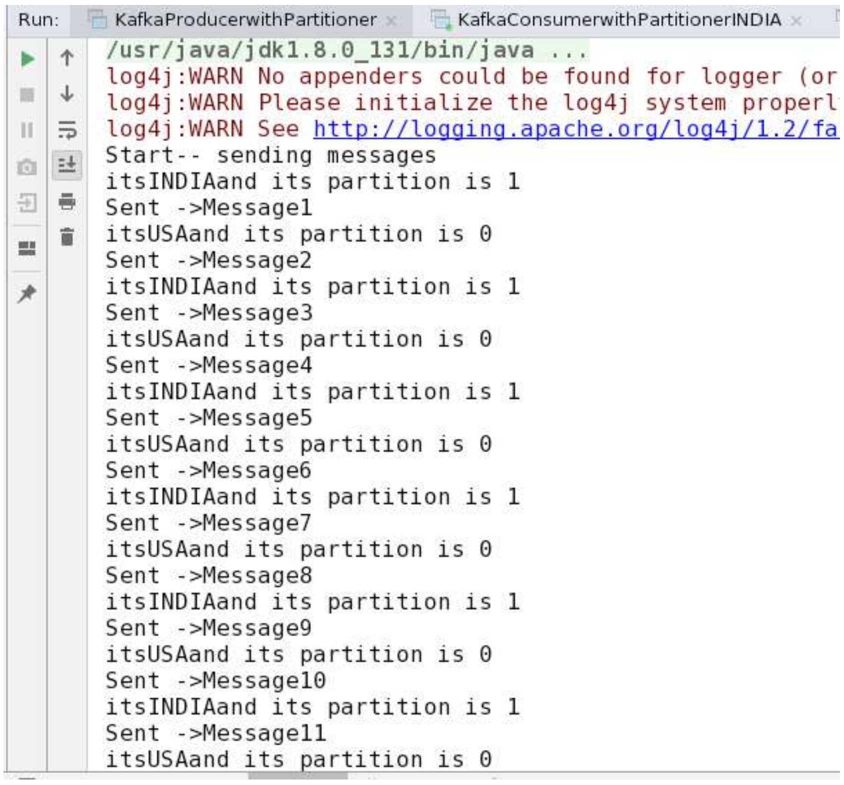
4. Let us create one partitioned consumer which will consume all messages from partition 1.

```
import org.apache.kafka.clients.consumer.ConsumerRecords

import org.apache.kafka.clients.consumer.KafkaConsumer
import Scala.util.{Collections, Properties}

import org.apache.kafka.common.TopicPartition


import scala.collection.ScalaConverters._


object KafkaConsumerwithPartitionerINDIA {

  def main(args: Array[String]): Unit = {


    val properties = new Properties()

    properties.put("bootstrap.servers", "localhost:9092")

    properties.put("group.id", "KafkaExampleNewConsumer2")

    properties.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer")

    properties.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer")

    properties.put("auto.offset.reset", "latest")

    val consumer = new KafkaConsumer[String, String](properties)


    //consumer.subscribe(util.Arrays.asList("testKafka"))

    val topicPartition = new TopicPartition("ipHits",1)

    consumer.assign(Collections.singletonList(topicPartition))


    while (true) {

      val records: ConsumerRecords[String, String] = consumer.poll(1000)

      records.asScala.foreach(record => println(s"Received message Partition:
"+record.partition() +"Offset:" +record.offset() +"Value "+record.value()))

    }


  }

}
```

5. Let us create one partitioned consumer which will consume all messages from partition 0.

```
import org.apache.kafka.clients.consumer._

import Scala.util.{Collections, Properties}

import org.apache.kafka.common.TopicPartition


import scala.collection.ScalaConverters._


object KafkaConsumerwithPartitionerUSA {

  def main(args: Array[String]): Unit = {

    val properties = new Properties()

    properties.put("bootstrap.servers", "localhost:9092")

    properties.put("group.id", "KafkaExampleNewConsumer")

    properties.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer")

    properties.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer")

    properties.put("auto.offset.reset", "latest")

    val consumer = new KafkaConsumer[String, String](properties)

    val topicPartition = new TopicPartition("ipHits",0)

    consumer.assign(Collections.singletonList(topicPartition))

      while (true) {

        val records: ConsumerRecords[String, String] = consumer.poll(1000)

        records.asScala.foreach(record => println(s"Received message Partition:
"+record.partition() +"Offset:" +record.offset() +"Value "+record.value()))

      }

    }

}
```

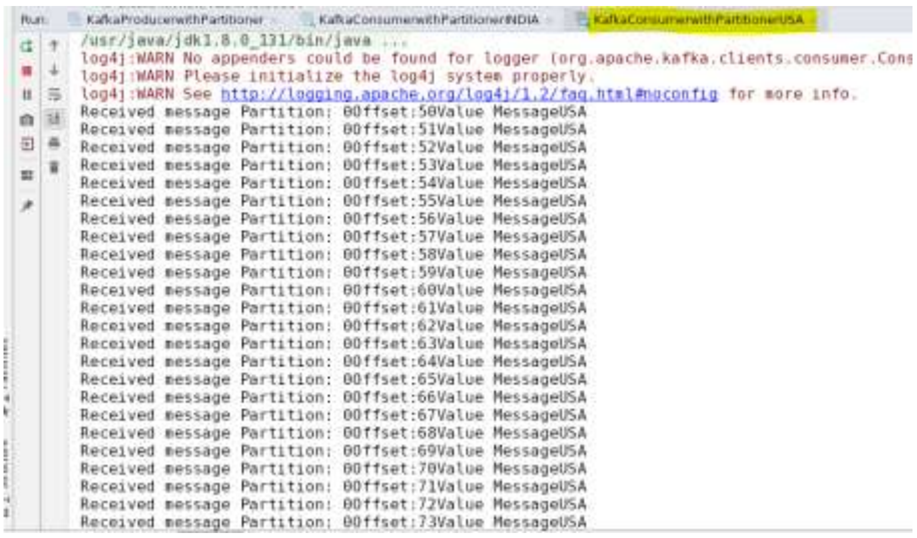
6. Start a producer as Scala Application.


Run:
KafkaProducerwithPartitioner
KafkaConsumerwithPartitionerINDIA

```
/usr/java/jdk1.8.0_131/bin/java ...
log4j:WARN No appenders could be found for logger (or
log4j:WARN Please initialize the log4j system properl
log4j:WARN See http://logging.apache.org/log4j/1.2/fa
Start-- sending messages
itsINDIAand its partition is 1
Sent ->Message1
itsUSAand its partition is 0
Sent ->Message2
itsINDIAand its partition is 1
Sent ->Message3
itsUSAand its partition is 0
Sent ->Message4
itsINDIAand its partition is 1
Sent ->Message5
itsUSAand its partition is 0
Sent ->Message6
itsINDIAand its partition is 1
Sent ->Message7
itsUSAand its partition is 0
Sent ->Message8
itsINDIAand its partition is 1
Sent ->Message9
itsUSAand its partition is 0
Sent ->Message10
itsINDIAand its partition is 1
Sent ->Message11
itsUSAand its partition is 0
```

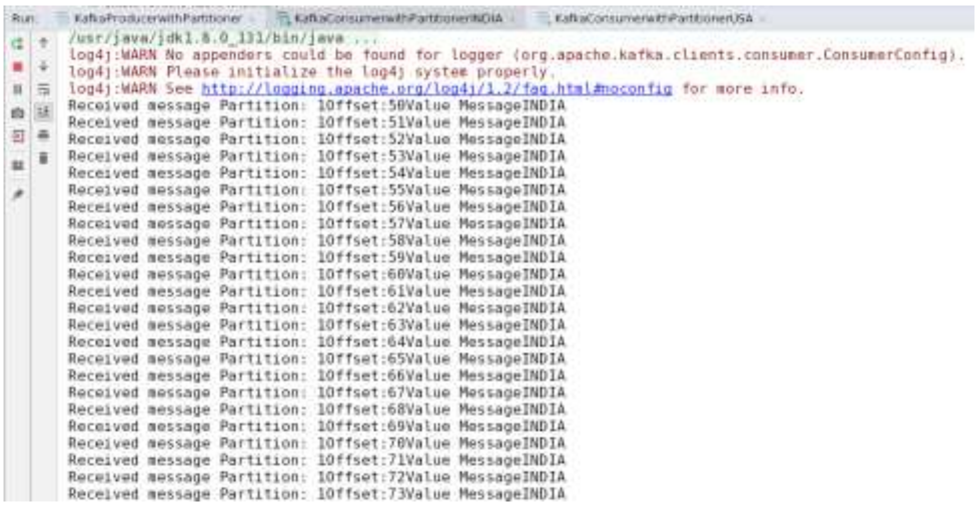
Note: Observe the value of the offset for message send to each partition.

7. Start the consumers as Scala Application.
   Navigate to <consumer-console>. Consumer for partition 0.


Run: KafkaProducerwithPartitioner
KafkaConsumerwithPartitionerINDIA
KafkaConsumerwithPartitionerUSA

```
/usr/java/jdk1.8.0_131/bin/java ...
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.consumer.Cons
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Received message Partition: 0Offset:50Value MessageUSA
Received message Partition: 0Offset:51Value MessageUSA
Received message Partition: 0Offset:52Value MessageUSA
Received message Partition: 0Offset:53Value MessageUSA
Received message Partition: 0Offset:54Value MessageUSA
Received message Partition: 0Offset:55Value MessageUSA
Received message Partition: 0Offset:56Value MessageUSA
Received message Partition: 0Offset:57Value MessageUSA
Received message Partition: 0Offset:58Value MessageUSA
Received message Partition: 0Offset:59Value MessageUSA
Received message Partition: 0Offset:60Value MessageUSA
Received message Partition: 0Offset:61Value MessageUSA
Received message Partition: 0Offset:62Value MessageUSA
Received message Partition: 0Offset:63Value MessageUSA
Received message Partition: 0Offset:64Value MessageUSA
Received message Partition: 0Offset:65Value MessageUSA
Received message Partition: 0Offset:66Value MessageUSA
Received message Partition: 0Offset:67Value MessageUSA
Received message Partition: 0Offset:68Value MessageUSA
Received message Partition: 0Offset:69Value MessageUSA
Received message Partition: 0Offset:70Value MessageUSA
Received message Partition: 0Offset:71Value MessageUSA
Received message Partition: 0Offset:72Value MessageUSA
Received message Partition: 0Offset:73Value MessageUSA
```

Consumer for partition 1.


Run: KafkaProducerwithPartitioner
KafkaConsumerwithPartitionerINDIA
KafkaConsumerwithPartitionerUSA

```
/usr/java/jdk1.8.0_131/bin/java ...
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.consumer.ConsumerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Received message Partition: 1Offset:50Value MessageINDIA
Received message Partition: 1Offset:51Value MessageINDIA
Received message Partition: 1Offset:52Value MessageINDIA
Received message Partition: 1Offset:53Value MessageINDIA
Received message Partition: 1Offset:54Value MessageINDIA
Received message Partition: 1Offset:55Value MessageINDIA
Received message Partition: 1Offset:56Value MessageINDIA
Received message Partition: 1Offset:57Value MessageINDIA
Received message Partition: 1Offset:58Value MessageINDIA
Received message Partition: 1Offset:59Value MessageINDIA
Received message Partition: 1Offset:60Value MessageINDIA
Received message Partition: 1Offset:61Value MessageINDIA
Received message Partition: 1Offset:62Value MessageINDIA
Received message Partition: 1Offset:63Value MessageINDIA
Received message Partition: 1Offset:64Value MessageINDIA
Received message Partition: 1Offset:65Value MessageINDIA
Received message Partition: 1Offset:66Value MessageINDIA
Received message Partition: 1Offset:67Value MessageINDIA
Received message Partition: 1Offset:68Value MessageINDIA
Received message Partition: 1Offset:69Value MessageINDIA
Received message Partition: 1Offset:70Value MessageINDIA
Received message Partition: 1Offset:71Value MessageINDIA
Received message Partition: 1Offset:72Value MessageINDIA
Received message Partition: 1Offset:73Value MessageINDIA
```

Exercise 17: Spark Streaming

Let’s start with simple streaming example, to count the number of words in text data received from a data server listening on a TCP socket.

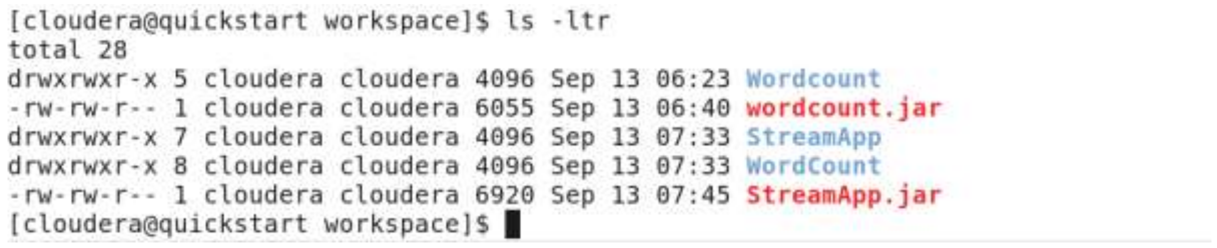
Create a DStream that represents streaming data from a TCP source, specified as hostname (e.g. localhost) and port (e.g. 9999).

Create a Scala project with below inputs:

```
import org.apache.spark.SparkConf

import org.apache.spark.SparkContext._

import org.apache.spark.streaming.StreamingContext

import org.apache.spark.streaming.Seconds

import org.apache.spark.storage.StorageLevel;

import org.apache.spark.streaming.StreamingContext._

object SparkStreamReceiver {

  def main(args: Array[String]): Unit = {

    var port:Int = 9999;

    val conf = new SparkConf()

                .setMaster("local[*]")

                .setAppName("Simple spark streaming")


    val ssc = new StreamingContext(conf, Seconds(3));

    val dstream = ssc.socketTextStream("localhost", port,
StorageLevel.MEMORY_ONLY);

    val words = dstream.flatMap(_.split(" "))

    val pairs = words.map(word => (word, 1))

    val wordcounts = pairs.reduceByKey(_ + _)

    wordcounts.print();

    ssc.start();

    ssc.awaitTermination();

  }

}
```

```
```

Export the jar file i.e. StreamApp.jar

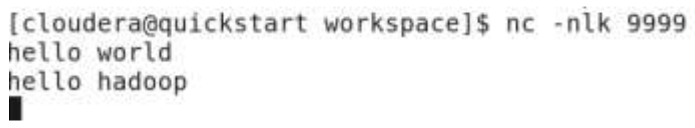


```
[cloudera@quickstart workspace]$ ls -ltr
total 28
drwxrwxr-x 5 cloudera cloudera 4096 Sep 13 06:23 Wordcount
-rw-rw-r-- 1 cloudera cloudera 6055 Sep 13 06:40 wordcount.jar
drwxrwxr-x 7 cloudera cloudera 4096 Sep 13 07:33 StreamApp
drwxrwxr-x 8 cloudera cloudera 4096 Sep 13 07:33 WordCount
-rw-rw-r-- 1 cloudera cloudera 6920 Sep 13 07:45 StreamApp.jar
[cloudera@quickstart workspace]$ 
```

Run the program using spark-submit

Run Netcat command and open another terminal to submit the job

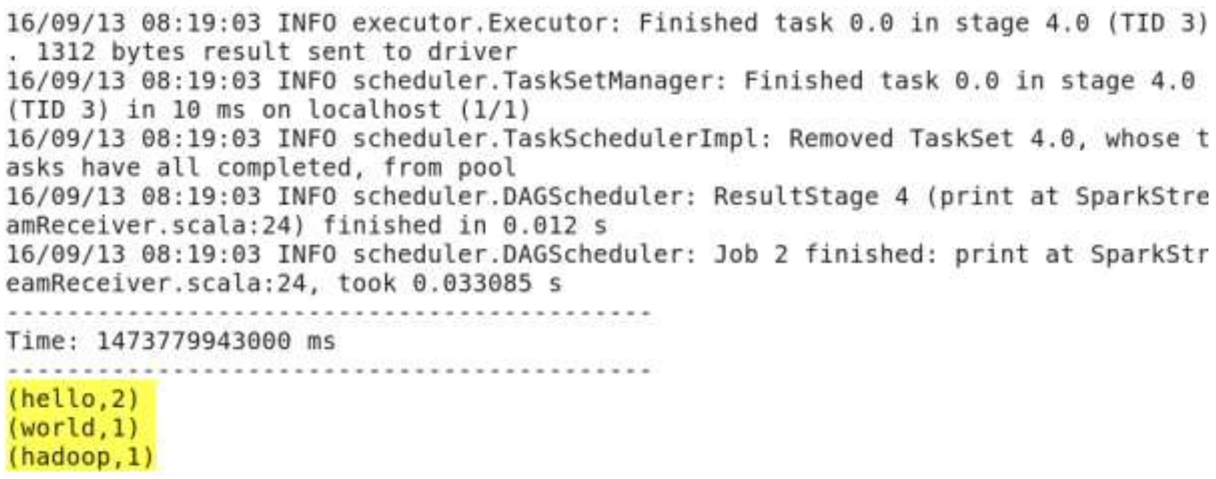
```
nc -nlk 9999
```



```
[cloudera@quickstart workspace]$ nc -nlk 9999
hello world
hello hadoop
```

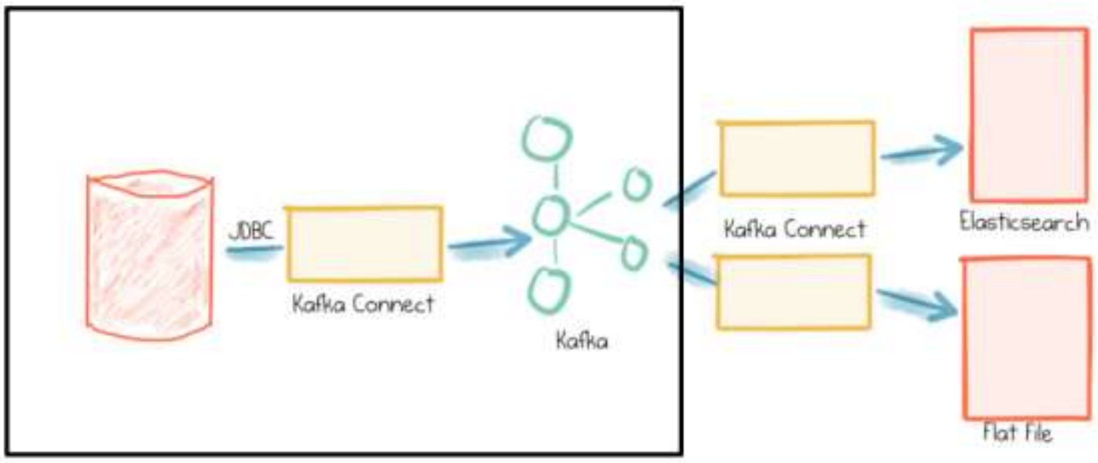
```
bin/spark-submit –master yarn --class solution.StreamingExample
/home/cloudera/eclipse-workspace/Lesson4/target/sfpdapp-1.0.jar
```

Any lines typed in the terminal running the netcat server will be counted and printed on screen every second. It will look something like the following



```
16/09/13 08:19:03 INFO executor.Executor: Finished task 0.0 in stage 4.0 (TID 3)
. 1312 bytes result sent to driver
16/09/13 08:19:03 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 4.0
(TID 3) in 10 ms on localhost (1/1)
16/09/13 08:19:03 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 4.0, whose t
asks have all completed, from pool
16/09/13 08:19:03 INFO scheduler.DAGScheduler: ResultStage 4 (print at SparkStre
amReceiver.scala:24) finished in 0.012 s
16/09/13 08:19:03 INFO scheduler.DAGScheduler: Job 2 finished: print at SparkStr
eamReceiver.scala:24, took 0.033085 s
-------------------------------------------
Time: 1473779943000 ms
-------------------------------------------
(hello,2)
(world,1)
(hadoop,1)
```

Exercise 18: Apache Kafka Connector Example – Import Data from MySQL into Kafka



We are going to take care of the black box, as surrounded in the diagram above.

The Confluent JDBC Connector for Kafka Connect enables you to stream data to and from Kafka and any RDBMS that supports JDBC (which is to say pretty much any). It can stream entire schemas or just individual tables.

Pre-requisites to work with Kafka connect

1. Confluent should be installed
2. Java version 1.8 should be installed
3. Mysql database should be installed
4. Kafka and Schema Registry are running locally on the default ports.

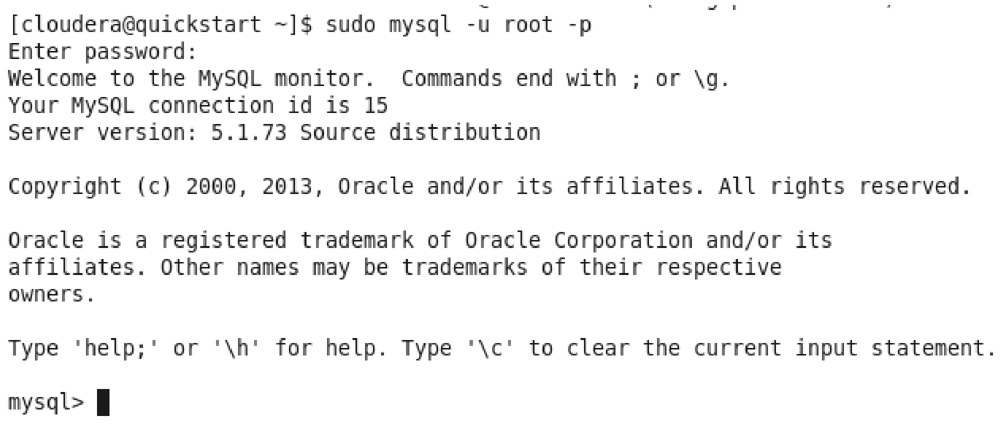
Steps to study the functionality of jdbc-connector using kafka connect

1. cd /home/cloudera/training/confluent-4.1.0

```
bin/confluent start schema-registry
```

2. Connect to MySQL with user root and password cloudera

```
[cloudera@quickstart ~]$ mysql -u root -p
```



```
[cloudera@quickstart ~]$ sudo mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 5.1.73 Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 
```

3. Next create a user, database, and a table:

```
mysql> create database kafkaconnect;

Query OK, 1 row affected (0.00 sec)
```

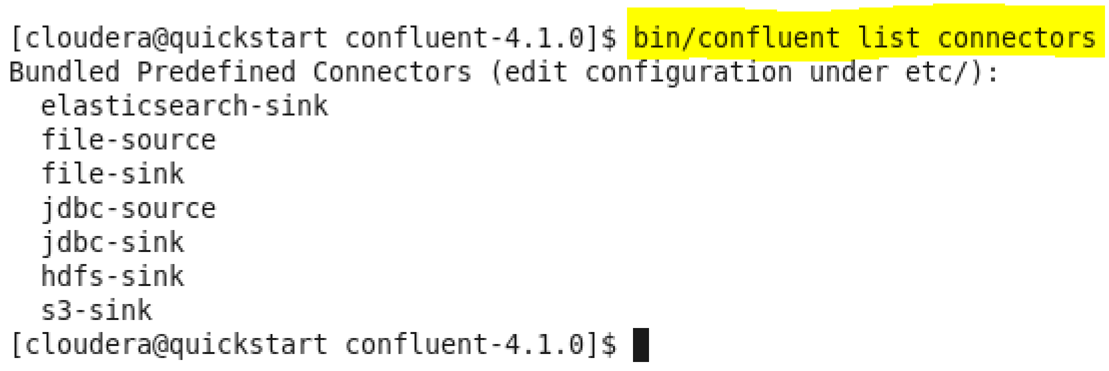
```
mysql> use kafkaconnect;

Database changed

mysql> CREATE TABLE authors(id INTEGER PRIMARY KEY

AUTO_INCREMENT NOT NULL, name VARCHAR(255));

mysql> INSERT INTO authors(name) VALUES('Manish');

mysql> INSERT INTO authors(name) VALUES('Chanchal');
```

Note: If the table exists in the database, drop the table.

Note: Please note that this will not work on MySQL versions earlier than 5.6

4. Loading the jdbc connector
   a) Checking the predefined connectors in confluent using the below command:
   b) Navigate to the location where confluent is installed.

```
bin/confluent list connectors
```



```
[cloudera@quickstart confluent-4.1.0]$ bin/confluent list connectors
Bundled Predefined Connectors (edit configuration under etc/):
  elasticsearch-sink
  file-source
  file-sink
  jdbc-source
  jdbc-sink
  hdfs-sink
  s3-sink
[cloudera@quickstart confluent-4.1.0]$ 
```

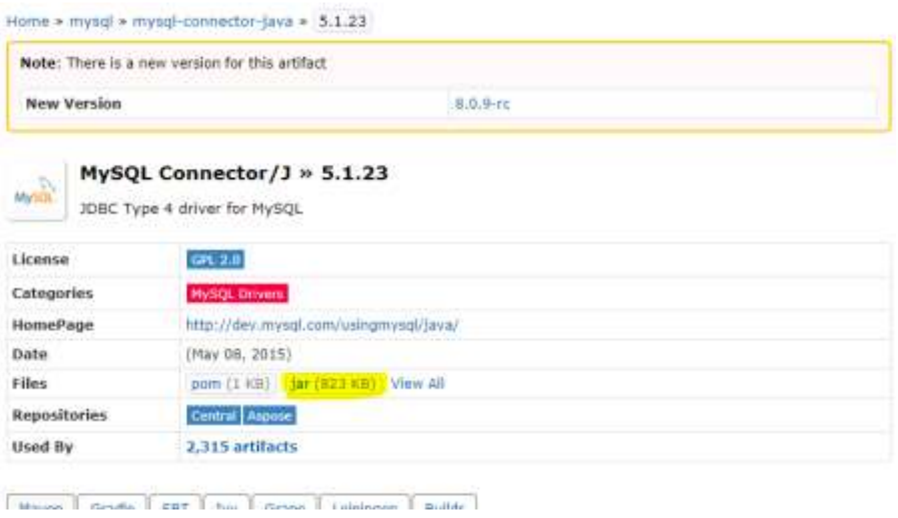
   c) Load the schema registry with the command to start schema registry on port 8081:
      from the location /home/cloudera/training/confluent-oss-4.0.0-2.11.tar/confluent-4.0.0 (navigate to the location)

```
bin/connect-standalone /home/cloudera/Desktop/SW/confluent-5.5.0/etc/schema-
registry/connect-avro-standalone.properties
/home/cloudera/Desktop/SW/confluent-5.5.0/etc/schema-registry/quickstart-
mysql.properties
```

```
(base) [cloudera@quickstart confluent-5.5.0]$ bin/connect-standalone /home/cloudera/Desktop/SW/confluent-5.5.0/etc/schema-registry/connect-avro-standalone.pro
perties /home/cloudera/Desktop/SW/confluent-5.5.0/etc/schema-registry/quickstart-mysql.properties
```

5. Download the mysql connector java jar from below location :
   Download the jar from https://mvnrepository.com/artifact/mysql/mysql-connector-java/5.1.23

   Click on the highlighted link “jar” which enables downloading the jar(yellow)



Home » mysql » mysql-connector-java » 5.1.23

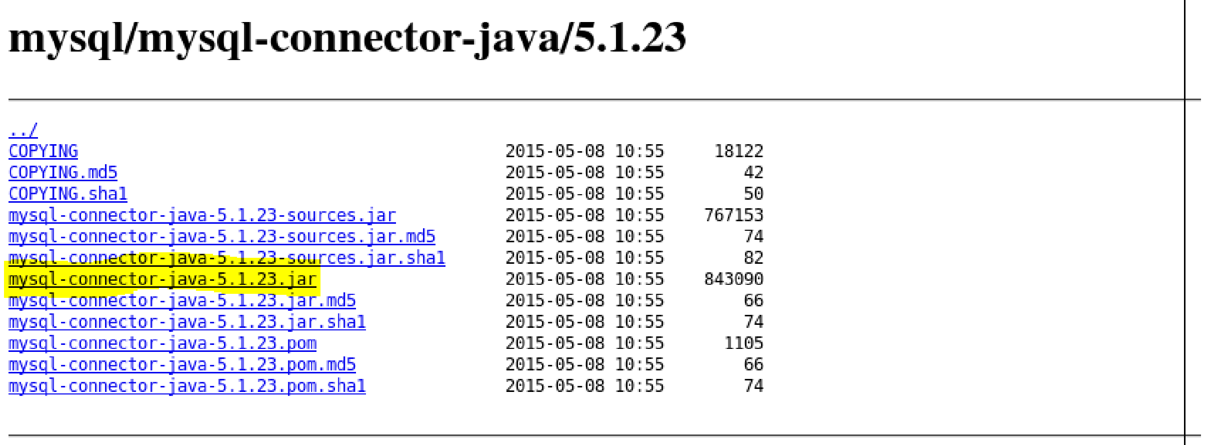
Note: There is a new version for this artifact

New Version	8.0.9-rc

MySQL Connector/J » 5.1.23

JDBC Type 4 driver for MySQL

License	GPL 2.0
Categories	MySQL Drivers
HomePage	http://dev.mysql.com/usingmysql/java/
Date	(May 08, 2015)
Files	pom (1 KB) jar (823 KB) View All
Repositories	Central Aspose
Used By	2,315 artifacts



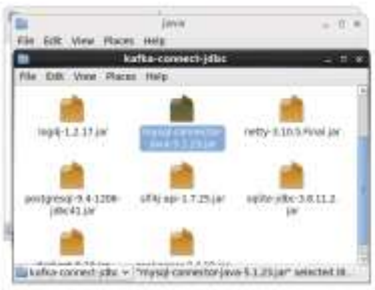
mysql/mysql-connector-java/5.1.23

```
../
COPYING                                           2015-05-08 10:55     18122
COPYING.md5                                       2015-05-08 10:55        42
COPYING.sha1                                      2015-05-08 10:55        50
mysql-connector-java-5.1.23-sources.jar           2015-05-08 10:55    767153
mysql-connector-java-5.1.23-sources.jar.md5       2015-05-08 10:55        74
mysql-connector-java-5.1.23-sources.jar.sha1      2015-05-08 10:55        82
mysql-connector-java-5.1.23.jar                   2015-05-08 10:55    843090
mysql-connector-java-5.1.23.jar.md5               2015-05-08 10:55        66
mysql-connector-java-5.1.23.jar.sha1              2015-05-08 10:55        74
mysql-connector-java-5.1.23.pom                   2015-05-08 10:55      1105
mysql-connector-java-5.1.23.pom.md5               2015-05-08 10:55        66
mysql-connector-java-5.1.23.pom.sha1              2015-05-08 10:55        74
```

Note: As part of the lab, the jar is already downloaded and is kept in the location.

6. Copy the above MySQL Connector Jar in the following location :

```
/home/cloudera/training/confluent-4.1.0/share/java/kafka-connect-jdbc
```


java
File Edit View Places Help
kafka-connect-jdbc
File Edit View Places Help
log4j-1.2.17.jar
mysql-connector-java-5.1.25.jar
netty-3.10.5.Final.jar
postgresql-9.4-1206-jdbc41.jar
slf4j-api-1.7.25.jar
sqlite-jdbc-3.8.11.2.jar
kafka-connect-jdbc
"mysql-connector-java-5.1.25.jar" selected

7. Create a quickstart-mysql.properties in the location </home/cloudera/training/confluent-4.1.0/share/java/kafka-connect-jdbc> with the below content:

```
name=test-mysql-jdbc-autoincrement

connector.class=io.confluent.connect.jdbc.JdbcSourceConnector

tasks.max=1

connection.url=jdbc:mysql://localhost:3306/kafkaconnect?user=root&password=cl
oudera

mode=incrementing

incrementing.column.name=id

topic.prefix=test-
```

Notes:
connection.url : In the connector configuration there are no security parameters. This is because SSL is not part of the JDBC standard and will depend on the JDBC driver in use. In general, you will need to configure SSL via the connection.url parameter which this jdbc driver connection url for mysql.

table.whitelist : is the configuration for setting the table which we want to copy in kafka

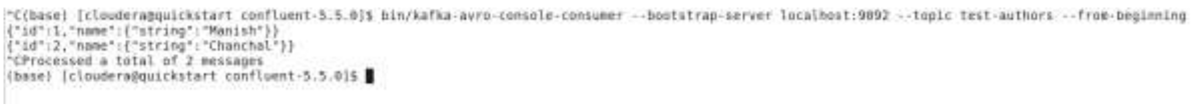
if autoincrementing column is set in a table , the column has to be specified.

topic-prefix : Prefix to prepend to table names to generate the name of the Kafka topic to publish data to.

```
[cloudera@quickstart confluent-4.1.0]$ bin/connect-standalone etc/schema-registry/connect-avro-standalone.properties /home/cloudera/training/confluent-4.1.0/share/java/kafka-connect-jdbc/quickstart-mysql.properties
[2018-10-01 11:58:23,584] INFO Kafka Connect standalone worker initializing ... (org.apache.kafka.connect.cli.ConnectStandalone:67)
[2018-10-01 11:58:23,522] INFO WorkerInfo values:
        jvm.args = [illegible]
        jvm.spec = Oracle Corporation, Java HotSpot(TM) 64-Bit Server VM, 1.8.0_131, 25.131-b11
        jvm.classpath = [illegible]
```

8. To check the data in kafka , run the Connect Avro console consumer

```
bin/kafka-avro-console-consumer --bootstrap-server localhost:9092 --topic test-authors --from-beginning
```



```
^C(base) [cloudera@quickstart confluent-5.5.0]$ bin/kafka-avro-console-consumer --bootstrap-server localhost:9092 --topic test-authors --from-beginning
{"id":1,"name":{"string":"Manish"}}
{"id":2,"name":{"string":"Chanchal"}}
^CProcessed a total of 2 messages
(base) [cloudera@quickstart confluent-5.5.0]$ █
```

9. Add another record via the mysql command prompt:

```
mysql INSERT INTO authors(name) VALUES('Rohan');
```

You can switch back to the console consumer and see the new record is added and, importantly, the old entries are not repeated:

Note: that the default polling interval is five seconds, so it may take a few seconds to show up. Depending on your expected rate of updates or desired latency, a smaller poll interval could be used to deliver updates more quickly.

Exercise 19: Apache Kafka Connector Example – Kafka Connect JDBC Sink

Prerequisites:

- Confluent should be installed
- Java version 1.8 should be installed
- Mysql database should be installed
- Kafka and Schema Registry are running locally on the default ports.
- MySQL connector is installed.

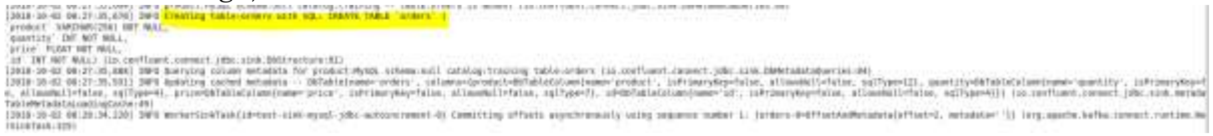
1. To Configure Data Sink Properties- Add a file as **sink-quickstart-mysql.properties** in the location etc/kafka-connect-jdbc.
   The content is

```
name=test-sink-mysql-jdbc-autoincrement

connector.class=io.confluent.connect.jdbc.JdbcSinkConnector

tasks.max=1

catalog=training

topics=orders

connection.url=jdbc:mysql://localhost:3306/training?user=root&password=clouder
a

auto.create=true
```

2. Let us Start standalone connector, with the command in terminal

```
bin/connect-standalone etc/schema-registry/connect-avro-standalone.properties
/home/cloudera/Desktop/SW/confluent-5.5.0/etc/schema-registry/sink-quickstart-
mysql.properties
```

   Observe the logs, the table is created



```
[2018-10-02 08:27:35,676] INFO Creating table:orders with SQL: CREATE TABLE `orders` (
`product` VARCHAR(256) NOT NULL,
`quantity` INT NOT NULL,
`price` FLOAT NOT NULL,
`id` INT NOT NULL) (io.confluent.connect.jdbc.sink.DbStructure:83)
[2018-10-02 08:27:35,886] INFO Querying column metadata for product:MySQL schema:null catalog:training table:orders (io.confluent.connect.jdbc.sink.DbMetadataQueries:44)
[2018-10-02 08:27:35,591] INFO Updating cached metadata -- DbTable{name='orders', columns={product=DbTableColumn{name='product', isPrimaryKey=false, allowsNull=false, sqlType=12}, quantity=DbTableColumn{name='quantity', isPrimaryKey=f
e, allowsNull=false, sqlType=4}, price=DbTableColumn{name='price', isPrimaryKey=false, allowsNull=false, sqlType=7}, id=DbTableColumn{name='id', isPrimaryKey=false, allowsNull=false, sqlType=4}}} (io.confluent.connect.jdbc.sink.metad
TableMetadataLoadingCache:49)
[2018-10-02 08:28:34,220] INFO WorkerSinkTask{id=test-sink-mysql-jdbc-autoincrement-0} Committing offsets asynchronously using sequence number 1: {orders-0=OffsetAndMetadata{offset=2, metadata=''}} (org.apache.kafka.connect.runtime.Wo
rkerSinkTask:325)
```

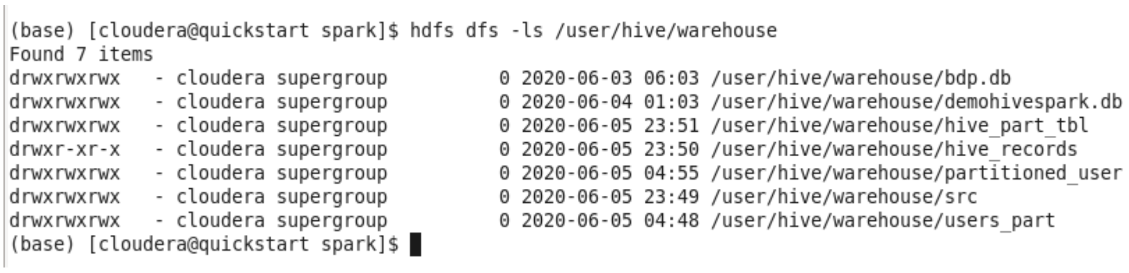
3. Let us Produce some record into the orders topic, passing the schema details of the table which will be automatically created

```
bin/kafka-avro-console-producer --broker-list localhost:9092 --topic orders --
property
value.schema='{"type":"record","name":"myrecord","fields":[{"name":"id","type":"in
t"},{"name":"product", "type": "string"}, {"name":"quantity", "type": "int"},
{"name":"price","type": "float"}]}'
```

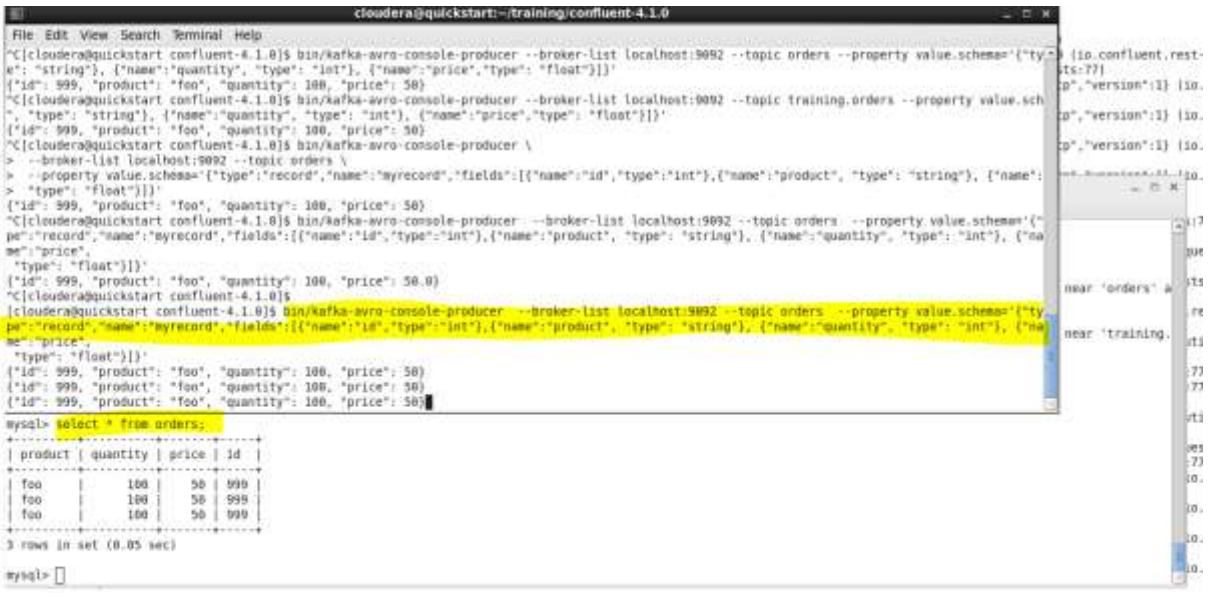
The console producer is waiting for input. Copy and paste the following record into the terminal and press **ENTER**

```
{"id": 999, "product": "foo", "quantity": 100, "price": 50}

{"id": 999, "product": "foo", "quantity": 100, "price": 50}

{"id": 999, "product": "foo", "quantity": 100, "price": 50}
```

4. Now if we query the database, we will see that the orders table was automatically created and contains the record.



```
(base) [cloudera@quickstart spark]$ hdfs dfs -ls /user/hive/warehouse
Found 7 items
drwxrwxrwx   - cloudera supergroup          0 2020-06-03 06:03 /user/hive/warehouse/bdp.db
drwxrwxrwx   - cloudera supergroup          0 2020-06-04 01:03 /user/hive/warehouse/demohivespark.db
drwxrwxrwx   - cloudera supergroup          0 2020-06-05 23:51 /user/hive/warehouse/hive_part_tbl
drwxr-xr-x   - cloudera supergroup          0 2020-06-05 23:50 /user/hive/warehouse/hive_records
drwxrwxrwx   - cloudera supergroup          0 2020-06-05 04:55 /user/hive/warehouse/partitioned_user
drwxrwxrwx   - cloudera supergroup          0 2020-06-05 23:49 /user/hive/warehouse/src
drwxrwxrwx   - cloudera supergroup          0 2020-06-05 04:48 /user/hive/warehouse/users_part
(base) [cloudera@quickstart spark]$ 
```



```
^C[cloudera@quickstart confluent-4.1.0]$ bin/kafka-avro-console-producer --broker-list localhost:9092 --topic orders --property value.schema='{"ty
e": "string"}, {"name":"quantity", "type": "int"}, {"name":"price","type": "float"}]}'
{"id": 999, "product": "foo", "quantity": 100, "price": 50}
^C[cloudera@quickstart confluent-4.1.0]$ bin/kafka-avro-console-producer --broker-list localhost:9092 --topic training.orders --property value.sch
", "type": "string"}, {"name":"quantity", "type": "int"}, {"name":"price","type": "float"}]}'
{"id": 999, "product": "foo", "quantity": 100, "price": 50}
^C[cloudera@quickstart confluent-4.1.0]$ bin/kafka-avro-console-producer \
>  --broker-list localhost:9092 --topic orders \
>  --property value.schema='{"type":"record","name":"myrecord","fields":[{"name":"id","type":"int"},{"name":"product", "type": "string"}, {"name":
>  "type": "float"}]}'
{"id": 999, "product": "foo", "quantity": 100, "price": 50}
^C[cloudera@quickstart confluent-4.1.0]$ bin/kafka-avro-console-producer  --broker-list localhost:9092 --topic orders  --property value.schema='{"
pe":"record","name":"myrecord","fields":[{"name":"id","type":"int"},{"name":"product", "type": "string"}, {"name":"quantity", "type": "int"}, {"na
me":"price",
 "type": "float"}]}'
{"id": 999, "product": "foo", "quantity": 100, "price": 50.0}
^C[cloudera@quickstart confluent-4.1.0]$
[cloudera@quickstart confluent-4.1.0]$ bin/kafka-avro-console-producer  --broker-list localhost:9092 --topic orders  --property value.schema='{"ty
pe":"record","name":"myrecord","fields":[{"name":"id","type":"int"},{"name":"product", "type": "string"}, {"name":"quantity", "type": "int"}, {"na
me":"price",
 "type": "float"}]}'
{"id": 999, "product": "foo", "quantity": 100, "price": 50}
{"id": 999, "product": "foo", "quantity": 100, "price": 50}
{"id": 999, "product": "foo", "quantity": 100, "price": 50}
```

```
mysql> select * from orders;
+---------+----------+-------+-----+
| product | quantity | price | id  |
+---------+----------+-------+-----+
| foo     |      100 |    50 | 999 |
| foo     |      100 |    50 | 999 |
| foo     |      100 |    50 | 999 |
+---------+----------+-------+-----+
3 rows in set (0.05 sec)

mysql> 
```

Exercise 20: Add Scala Kernal in Jupyter

Open a terminal and navigate to root folder

```
cd ~
```

➢ Download almond and scala libs

```
$ curl -Lo coursier https://git.io/coursier-cli && chmod +x coursier
```

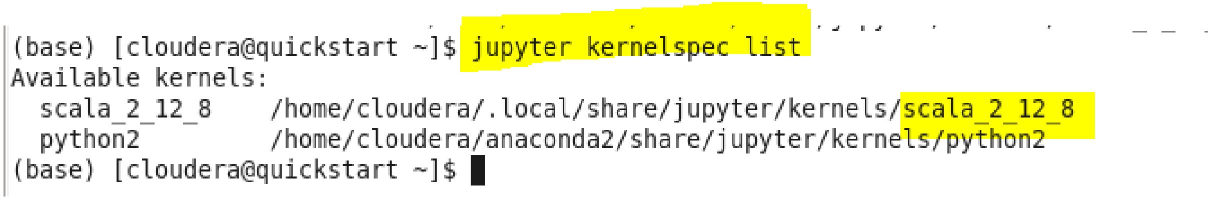
```
$ ./coursier bootstrap \
  -r jitpack \
  -i user -I user:sh.almond:scala-kernel-api_2.12.8:0.7.0 \
  sh.almond:scala-kernel_2.12.8:0.7.0 \
-o almond
```

➢ Add Scala kernel to Jupyter

```
$ ./almond --install --id scala_2_12_8  --display-name "Scala 2.12.8"
```

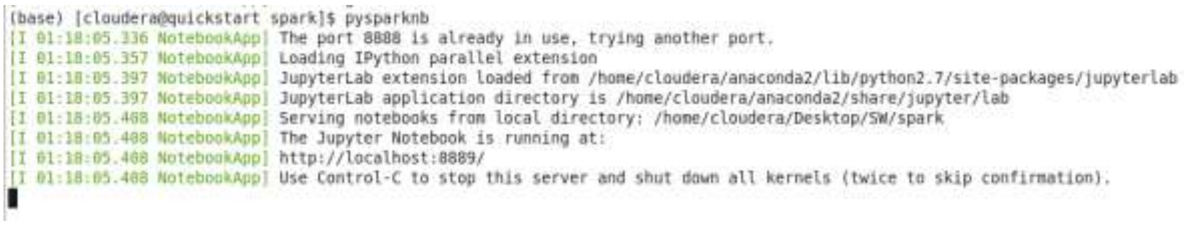
➢ List kernelsPermalink

```
jupyter kernelspec list
```



```
(base) [cloudera@quickstart ~]$ jupyter kernelspec list
Available kernels:
  scala_2_12_8    /home/cloudera/.local/share/jupyter/kernels/scala_2_12_8
  python2         /home/cloudera/anaconda2/share/jupyter/kernels/python2
(base) [cloudera@quickstart ~]$
```

Launch pysparknb to launch jupyter notebook from the terminal.

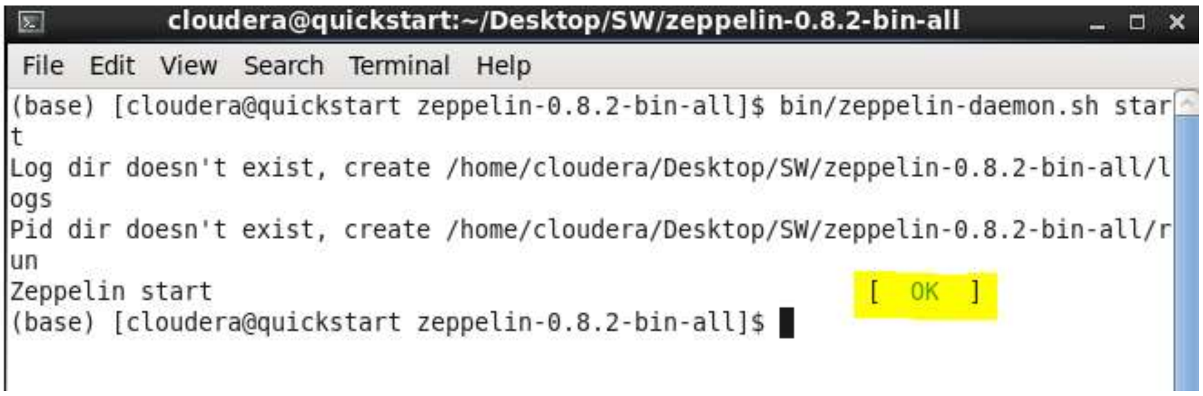


```
(base) [cloudera@quickstart spark]$ pysparknb
[I 01:18:05.336 NotebookApp] The port 8888 is already in use, trying another port.
[I 01:18:05.357 NotebookApp] Loading IPython parallel extension
[I 01:18:05.397 NotebookApp] JupyterLab extension loaded from /home/cloudera/anaconda2/lib/python2.7/site-packages/jupyterlab
[I 01:18:05.397 NotebookApp] JupyterLab application directory is /home/cloudera/anaconda2/share/jupyter/lab
[I 01:18:05.408 NotebookApp] Serving notebooks from local directory: /home/cloudera/Desktop/SW/spark
[I 01:18:05.408 NotebookApp] The Jupyter Notebook is running at:
[I 01:18:05.408 NotebookApp] http://localhost:8889/
[I 01:18:05.408 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

Its running, open mozilla with the provided url.

To run Zeppelin and create a Zeppelin notebook, run the following command :

zeppelin-daemon.sh start


cloudera@quickstart:~/Desktop/SW/zeppelin-0.8.2-bin-all
File Edit View Search Terminal Help

```
(base) [cloudera@quickstart zeppelin-0.8.2-bin-all]$ bin/zeppelin-daemon.sh star
t
Log dir doesn't exist, create /home/cloudera/Desktop/SW/zeppelin-0.8.2-bin-all/l
ogs
Pid dir doesn't exist, create /home/cloudera/Desktop/SW/zeppelin-0.8.2-bin-all/r
un
Zeppelin start                                             [  OK  ]
(base) [cloudera@quickstart zeppelin-0.8.2-bin-all]$
```

There are some useful Zeppelin commands one should know :

$ zeppelin-daemon.sh start -> To start the Daemon

$ zeppelin-daemon.sh stop -> To stop the Daemon

$ zeppelin-daemon.sh restart -> To restart the Daemon

Your Zeppelin Notebook should be accessible from the following link : http://localhost:8080/