



# Cloudera

## Exam CCA175


CCA Spark and Hadoop Developer Exam

Verson: Demo

[ Total Questions: 10 ]



**PASS YOUR EXAM  
IN FIRST ATTEMPT  
REAL EXAM QUESTIONS**



### Question No : 1 CORRECT TEXT

Problem Scenario 22 : You have been given below comma separated employee information.

name,salary,sex,age

alok,100000,male,29

jatin,105000,male,32

yogesh,134000,male,39

ragini,112000,female,35

jjyotsana,129000,female,39

valmiki,123000,male,29

Use the netcat service on port 44444, and nc above data line by line. Please do the following activities.

1. Create a flume conf **file using fastest** channel, which write data **in** hive warehouse directory, in **a table called** flumeemployee (Create hive table as well for given data).
2. Write a hive query to read average salary of all employees.

**Answer:** See the explanation for Step by Step Solution and configuration.

#### **Explanation:**

Solution :

Step 1 : **Create hive table** for flumeemployee.'

```
CREATE TABLE flumeemployee
```

```
(
```

```
name string, salary int, sex string,
```

```
age int
```

```
)
```

```
ROW FORMAT DELIMITED
```

```
FIELDS TERMINATED BY ',';
```

Step 2 : Create flume configuration file, with below configuration for source, sink **and**

---

channel and save it in flume2.conf.

#Define source , sink , channel and agent,

agent1.sources = source1

agent1.sinks = sink1

agent1.channels = channel1

# Describe/configure source1

agent1.sources.source1.type = netcat

agent1.sources.source1.bind = 127.0.0.1

agent1.sources.source1.port = 44444

## Describe sink1

agent1.sinks.sink1.channel = memory-channel

agent1.sinks.sink1.type = hdfs

agent1.sinks.sink1.hdfs.path = /user/hive/warehouse/flumeemployee

hdfs-agent.sinks.hdfs-write.hdfs.writeFormat=Text

agent1.sinks.sink1.hdfs.tileType = Data Stream

# Now we need to define channel1 property.

agent1.channels.channel1.type = memory

agent1.channels.channel1.capacity = 1000

agent1.channels.channel1.transactionCapacity = 100

# Bind the source and sink to the channel

Agent1.sources.source1.channels = channel1 agent1.sinks.sink1.channel = channel1

Step 3 : Run below command which **will use this** configuration file and append data in hdfs.

Start flume service:

```
flume-ng agent -conf /home/cloudera/flumeconf -conf-file  
/home/cloudera/flumeconf/flume2.conf --name agent1
```

Step 4 : Open another terminal and use the netcat service.

nc localhost 44444

Step 5 : Enter data line by line.

alok,100000,male,29

jatin,105000,male,32

yogesh,134000,male,39

ragini,112000,female,35

---

jyotsana,129000,female,39  
 valmiki,123000,male,29

Step 6 : Open hue and check the data is **available in hive** table or not.

step 7 : Stop flume service by pressing ctrl+c

Step 8 : Calculate average salary on hive table using below query. You can use either hive command line tool or hue. select avg(salary) from flumeemployee;

### Question No : 2 CORRECT TEXT

Problem Scenario 50 : You have been given below code snippet (calculating an average score}, with intermediate output.

```
type ScoreCollector = (Int, Double)
```

```
type PersonScores = (String, (Int, Double))
```

```
val initialScores = Array(("Fred", 88.0), ("Fred", 95.0), ("Fred", 91.0), ("Wilma", 93.0),  
 ("Wilma", 95.0), ("Wilma", 98.0))
```

```
val wilmaAndFredScores = sc.parallelize(initialScores).cache()
```

```
val scores = wilmaAndFredScores.combineByKey(createScoreCombiner, scoreCombiner,  
 scoreMerger)
```

```
val averagingFunction = (personScore: PersonScores) => { val (name, (numberScores,  
 totalScore)) = personScore (name, totalScore / numberScores)
```

```
}
```

```
val averageScores = scores.collectAsMap().map(averagingFunction)
```

Expected output: averageScores: scala.collection.Map[String,Double] = Map(Fred -> 91.33333333333333, Wilma -> 95.33333333333333)

Define all three required **function , which are input for** combineByKey method, e.g. (createScoreCombiner, scoreCombiner, scoreMerger). And help us producing required results.

---

**Answer:** See the explanation for Step by Step Solution and configuration.

**Explanation:**

Solution :

```
val createScoreCombiner = (score: Double) => (1, score)
val scoreCombiner = (collector: ScoreCollector, score: Double) => {
val (numberScores, totalScore) = collector (numberScores + 1, totalScore + score)
}
val scoreMerger= (collector-!: ScoreCollector, collector2: ScoreCollector) => { val
(numScores1, totalScore1) = collector! val (numScores2, totalScore2) = collector
(numScores1 + numScores2, totalScore1 + totalScore2)
}
```

**Question No : 3 CORRECT TEXT**

Problem Scenario 42 : You have been given a file (spark10/sales.txt), with the content **as given in below.**

**spark10/sales.txt**

**Department,Designation,costToCompany,State**

**Sales,Trainee,12000,UP**

**Sales,Lead,32000,AP**

**Sales,Lead,32000,LA**

**Sales,Lead,32000,TN**

**Sales,Lead,32000,AP**

**Sales,Lead,32000,TN**

**Sales,Lead,32000,LA**

**Sales,Lead,32000,LA**

**Marketing,Associate,18000,TN**

**Marketing,Associate,18000,TN**

**HR,Manager,58000,TN**

---

**And** want to produce the output **as a csv** with group **by Department,Designation,State** with additional **columns** with sum(costToCompany) and TotalEmployeeCountt

Should get result like

Dept,Desg,state,empCount,totalCost

Sales,Lead,AP,2,64000

Sales.Lead.LA.3.96000

Sales,Lead,TN,2,64000

**Answer:** See the explanation for Step by Step Solution and configuration.

**Explanation:**

Solution :

step 1 : **Create a file first using Hue** in hdfs.

Step 2 : Load file as an RDD

```
val rawlines = sc.textFile("spark10/sales.txt")
```

Step 3 : Create a case class, which can represent its column fields. case class

```
Employee(dep: String, des: String, cost: Double, state: String)
```

Step 4 : Split the data **and** create RDD of all Employee objects.

```
val employees = rawlines.map(_._split(",")).map(row=>Employee(row(0), row{1},  
row{2}.toDouble, row{3}))
```

Step 5 : Create a row as we needed. All group by fields as a key and value as a count for each employee as well as its cost, val keyVals = employees.map( em => ((em.dep, em.des, em.state), (1 , em.cost)))

Step 6 : Group by all the records using reduceByKey method as we want summation as well. For number of employees and their total cost, val results = keyVals.reduceByKey{(a,b) => (a.\_1 + b.\_1, a.\_2 + b.\_2)} // (a.count + b.count, a.cost + b.cost)}

Step 7 : Save the results in a text file as below.

```
results.repartition(1).saveAsTextFile("spark10/group.txt")
```

---

**Question No : 4 CORRECT TEXT**

Problem Scenario 21 : You have been given log generating service as below.

startjogs (It will generate continuous logs)

tailjogs (You can check , what logs are being generated)

stopjogs (It will stop the log service)

Path where logs are generated using above service : /opt/gen\_logs/logs/access.log

Now write a flume configuration file named flumel.conf , using that configuration file dumps logs in HDFS file system in a directory called flumel. Flume channel should have following property as well. After every 100 message it should be committed, use non-durable/faster channel and it should be able to hold maximum 1000 events

Solution :

Step 1 : Create flume configuration file, with below configuration for source, sink and channel.

#Define source , sink , channel and agent,

agent1.sources = source1

agent1.sinks = sink1

agent1.channels = channel1

# Describe/configure source1

agent1.sources.source1.type = exec

agent1.sources.source1.command = tail -F /opt/gen\_logs/logs/access.log

## Describe sink1

agent1.sinks.sink1.channel = memory-channel

agent1.sinks.sink1.type = hdfs

agent1.sinks.sink1.hdfs.path = flumel

---

```
agent1.sinks.sink1.hdfs.fileType = Data Stream
```

```
# Now we need to define channel property.
```

```
agent1.channels.channel1.type = memory
```

```
agent1.channels.channel1.capacity = 1000
```

```
agent1.channels.channel1.transactionCapacity = 100
```

```
# Bind the source and sink to the channel
```

```
agent1.sources.source1.channels = channel1
```

```
agent1.sinks.sink1.channel = channel1
```

Step 2 : Run below command which **will use this** configuration file and append data in hdfs.

Start log service using : startjogs

Start flume service:

```
flume-ng agent -conf /home/cloudera/flumeconf -conf-file  
/home/cloudera/flumeconf/flume1.conf-Dflume.root.logger=DEBUG,INFO,console
```

Wait for few mins and then stop log service.

Stop\_logs

**Answer:** See the explanation for Step by Step Solution and configuration.

#### **Question No : 5 CORRECT TEXT**

Problem Scenario 68 : You have given a file as below.

```
spark75/file1.txt
```

**File contain** some text. As given Below

```
spark75/file1.txt
```



---

Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common and should be automatically handled by the framework

The core of Apache Hadoop consists of a storage part known as Hadoop Distributed File System (HDFS) and a processing part called MapReduce. Hadoop splits files into large blocks and distributes them across nodes in a cluster. To process data, Hadoop transfers packaged code for nodes to process in parallel based on the data that needs to be processed.

his approach takes advantage of data locality nodes manipulating the data they have access to to allow the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking

For a slightly more complicated task, lets look into splitting up sentences from our documents into word bigrams. A bigram is pair of successive tokens in some sequence. We will look at building bigrams from the sequences of words in each sentence, and then try to find the most frequently occurring ones.

The first problem is that values in each partition of our initial RDD describe lines from the file rather than sentences. Sentences may be split over multiple lines. The `glom()` RDD method is used to create a single entry for each document containing the list of all lines, we can then join the lines up, then resplit them into sentences using "." as the separator, using `flatMap` so that every object in our RDD is now a sentence.

A bigram is pair of successive tokens in some sequence. Please build bigrams from the sequences of words in each sentence, and then try to find the most frequently occurring ones.

**Answer:** See the explanation for Step by Step Solution and configuration.

**Explanation:**

Solution :

Step 1 : Create all three tiles in hdfs (We will do using Hue}. However, you can first create in local filesystem and then upload it to hdfs.

Step 2 : The first problem is that values in each partition of our initial RDD describe lines from the file rather than sentences. Sentences may be split over multiple lines.

The `glom()` **RDD** method is used to create a single entry for each document containing the

---

list of all lines, we can then join the lines up, then resplit them into sentences using "." as the separator, using flatMap so that every object in our RDD is now a sentence.

```
sentences = sc.textFile("spark75/file1.txt") \ .glom() \  
map(lambda x: " ".join(x)) \ .flatMap(lambda x: x.split("."))
```

Step 3 : Now we have isolated each sentence we can split it into a list of words and extract the word bigrams from it. Our new RDD contains tuples containing the word bigram (itself a tuple containing the first and second word) as the first value and the number 1 as the second value. bigrams = sentences.map(lambda x:x.split()) \ .flatMap(lambda x: [((x[i],x[i+1]),1)for i in range(0,len(x)-1)])

Step 4 : Finally we can apply the same reduceByKey and sort steps that we used in the wordcount example, to count up the bigrams and sort them in order of descending frequency. In reduceByKey the key is **not an individual** word but a bigram.

```
freq_bigrams = bigrams.reduceByKey(lambda x,y:x+y)\  
map(lambda x:(x[1],x[0])) \  
sortByKey(False)  
freq_bigrams.take(10)
```

### Question No : 6 CORRECT TEXT

Problem Scenario 32 : You have given three files as below.

spark3/sparkdir1/file1.txt

spark3/sparkdir2/file2.txt

spark3/sparkdir3/file3.txt

Each file contain some text.

#### spark3/sparkdir1/file1.txt

Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common and should be automatically handled by the framework

---

**spark3/sparkdir2/file2.txt**

The core of Apache Hadoop consists of a storage part known as Hadoop Distributed File System (HDFS) and a processing part called MapReduce. Hadoop splits files into large blocks and distributes them across nodes in a cluster. To process data, Hadoop transfers packaged code for nodes to process in parallel based on the data that needs to be processed.

**spark3/sparkdir3/file3.txt**

his approach takes advantage of data locality nodes manipulating the data they have access to to allow the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking

Now write a Spark code in scala which will load all these three files from hdfs and do the word count by filtering following words. And result should be sorted by word count in reverse order.

Filter words ("a","the","an", "as", "a","with","this","these","is","are","in", "for", "to","and","The","of")

Also please make sure you load all three files as a Single RDD (All three files must be loaded using single API call).

You have also been given following codec

```
import org.apache.hadoop.io.compress.GzipCodec
```

Please use above codec to compress file, while saving in hdfs.

**Answer:** See the explanation for Step by Step Solution and configuration.

**Explanation:**

Solution :

**Step 1 :** Create all three files in hdfs (We will do using Hue). However, you can first create in local filesystem and then upload it to hdfs.

**Step 2 :** Load content from all files.

---

```
val content =  
sc.textFile("spark3/sparkdir1/file1.txt,spark3/sparkdir2/file2.txt,spark3/sparkdir3/file3.  
txt") //Load the text file
```

**Step 3 : Now create** split each line and **create** RDD of words.

```
val flatContent = content.flatMap(word=>word.split(" "))
```

step 4 : Remove space after each word (trim it)

```
val trimmedContent = flatContent.map(word=>word.trim)
```

Step 5 : Create an RDD from remove, all the words that needs to be removed.

```
val removeRDD = sc.parallelize(List("a","theM,ManM, "as",  
"a","with","this","these","is","are\\in\\ "for", "to","and","The","of"))
```

Step 6 : Filter the RDD, so it can have only content which are not present in removeRDD.

```
val filtered = trimmedContent.subtract(removeRDD)
```

Step 7 : Create a PairRDD, so we can have (word,1) tuple or PairRDD. val pairRDD =  
filtered.map(word => (word,1))

Step 8 : Now do the word count on PairRDD. val wordCount = pairRDD.reduceByKey(\_ +  
\_)

Step 9 : Now swap PairRDD.

```
val swapped = wordCount.map(item => item.swap)
```

Step 10 : Now revers order the content. val sortedOutput = swapped.sortByKey(false)

Step 11 : Save the output as a Text file. sortedOutput.saveAsTextFile("spark3/result")

Step 12 : Save compressed output.

```
import org.apache.hadoop.io.compress.GzipCodec  
sortedOutput.saveAsTextFile("spark3/compressedresult", classOf[GzipCodec])
```

**Question No : 7 CORRECT TEXT**

---

Problem Scenario 34 : You have given a file named spark6/user.csv.

Data is given below:

user.csv

id,topic,hits

Rahul,scala,120

Nikita,spark,80

Mithun,spark,1

myself,cca175,180

Now write a Spark code in scala which will remove the header part and create RDD of values as below, for all rows. And also if id is myself" than filter out row.

Map(id -> om, topic -> scala, hits -> 120)

**Answer:** See the explanation for Step by Step Solution and configuration.

**Explanation:**

Solution :

Step 1 : Create file in hdfs (We will do using Hue). However, you can first create in local filesystem and then upload it to hdfs.

Step 2 : Load user.csv file from hdfs and create PairRDDs val csv =  
sc.textFile("spark6/user.csv")

Step 3 : split and clean data

```
val headerAndRows = csv.map(line => line.split(",").map(_.trim))
```

Step 4 : Get header row

```
val header = headerAndRows.first
```

Step 5 : Filter out header (We need to check if the first val matches the first header name)

```
val data = headerAndRows.filter(_(0) != header(0))
```

Step 6 : Splits to map (header/value pairs)

```
val maps = data.map(splits => header.zip(splits).toMap)
```

---

**step 7: Filter out the user "myself"**

```
val result = maps.filter(map => map["id"] != "myself")
```

Step 8 : Save the output as a Text file. `result.saveAsTextFile("spark6/result.txt")`

**Question No : 8 CORRECT TEXT**

Problem Scenario 11 : You have been given following mysql database details as well as other info.

user=retail\_dba

password=cloudera

database=retail\_db

jdbc URL = jdbc:mysql://quickstart:3306/retail\_db

**Please accomplish following.**

**1. Import departments table in a directory called departments.**

**2. Once import is done, please insert following 5 records in departments mysql table.**

**Insert into departments(10, physics);**

**Insert into departments(11, Chemistry);**

**Insert into departments(12, Maths);**

**Insert into departments(13, Science);**

**Insert into departments(14, Engineering);**

**3. Now import only new inserted records and append to existing directory . which has been created in first step.**

**Answer:** See the explanation for Step by Step Solution and configuration.

---

**Explanation:**

Solution :

Step 1 : Clean already imported data. (In real exam, please make sure you dont delete data generated from previous exercise).

```
hadoop fs -rm -R departments
```

Step 2 : Import data in departments directory.

```
sqoop import \  
--connect jdbc:mysql://quickstart:3306/retail_db \  
--username=retail_dba \  
-password=cloudera \  
-table departments \  
"target-dir/user/cloudera/departments"
```

Step 3 : Insert the five records in departments table.

```
mysql -user=retail_dba --password=cloudera retail_db  
Insert into departments values(10, "physics"); Insert into departments values(11,  
"Chemistry"); Insert into departments values(12, "Maths"); Insert into departments  
values(13, "Science"); Insert into departments values(14, "Engineering"); commit;  
select' from departments;
```

Step 4 : Get the maximum value of departments from last import, `hdfs dfs -cat /user/cloudera/departments/part*` that should be 7

Step 5 : Do the incremental import based on last import and append the results.

```
sqoop import \  
--connect "jdbc:mysql://quickstart.cloudera:330G/retail_db" \  
~username=retail_dba \  
-password=cloudera \  
-table departments \  
--target-dir /user/cloudera/departments \  
-append \  
-check-column "department_id" \  
-incremental append \  
-last-value 7
```

Step 6 : Now check the result.

```
hdfs dfs -cat /user/cloudera/departments/part"
```

---

**Question No : 9 CORRECT TEXT**

Problem Scenario 95 : You have to run your Spark application on yarn with each executor Maximum heap size to be 512MB and Number of processor cores to allocate on each executor will be 1 and Your main application required three values as input arguments V1 V2 V3.

Please replace XXX, YYY, ZZZ

```
./bin/spark-submit -class com.hadoopexam.MyTask --master yarn-cluster--num-executors 3  
--driver-memory 512m XXX YYY lib/hadoopexam.jarZZZ
```

**Answer:** See the explanation for Step by Step Solution and configuration.

**Explanation:**

Solution

XXX: -executor-memory 512m YYY: -executor-cores 1

ZZZ : V1 V2 V3

Notes : spark-submit on yarn options Option Description

archives Comma-separated list of archives to be extracted into the working directory of each executor. The path must be globally visible inside your cluster; see Advanced Dependency Management.

executor-cores Number of processor cores to allocate on each executor. Alternatively, you can use the spark.executor.cores property, executor-memory Maximum heap size to allocate to each executor. Alternatively, you can use the spark.executor.memory-property.

num-executors Total number of YARN containers to allocate for this application.

Alternatively, you can use the spark.executor.instances property. queue YARN queue to submit to. For more information, see Assigning Applications and Queries to Resource Pools. Default: default.

---

**Question No : 10 CORRECT TEXT**

Problem Scenario 92 : You have been given a spark scala application, which is bundled in



---

jar named hadoopexam.jar.

Your application class name is com.hadoopexam.MyTask

You want that while submitting your application should launch a driver on one of the cluster node.

Please complete the following command to submit the application.

spark-submit XXX -master yarn \

YYY SSPARK HOME/lib/hadoopexam.jar 10

**Answer:** See the explanation for Step by Step Solution and configuration.

**Explanation:**

Solution

XXX: -class com.hadoopexam.MyTask

YYY : --deploy-mode cluster