Quick reference table for reading and writing into several file formats in hdfs

File Format	Action	Procedure and points to remember
TEXT FILE	READ	sparkContext.textFile(<path to file>);
	WRITE	sparkContext.saveAsTextFile(<path to file>,classOf[**compressionCodecClass**]); //use any codec here org.apache.hadoop.io.compress.(**BZip2Codec** or GZipCodec or SnappyCodec)
SEQUENCE FILE	READ	sparkContext.sequenceFile(<path location>,classOf[<class name>],classOf[<**compressionCodecClass** >]); //read the head of sequence file to understand what two class names need to be used here

	WRITE	rdd.saveAsSequenceFile(<path location>, Some(classOf[compressionCodecClass]))
		//use any codec here (**BZip2Codec**,GZipCodec,SnappyCodec)
		//here rdd is MapPartitionRDD and not the regular pair RDD.

PARQUET FILE	READ	//use data frame to load the file. sqlContext.read.parquet(<path to location>); //this results in a data frame object.

	WRITE	sqlContext.setConf("spark.sql.parquet.compression.codec","gzip")
		//use gzip, snappy, lzo or uncompressed here
		dataFrame.write.parquet(<path to location>);

ORC FILE	READ	sqlContext.read.orc(<path to location>); //this results in a dataframe
	WRITE	df.write.mode(SaveMode.Overwrite).format("orc") .save(<path to location>)
AVRO FILE	READ	import com.databricks.spark.avro._; sqlContext.read.avro(<path to location>); // this results in a data frame object

	WRITE	sqlContext.setConf("spark.sql.avro.compression.codec","snappy")
		//use snappy, deflate, uncompressed;
		dataFrame.write.avro(<path to location>);

JSON FILE	READ	sqlContext.read.json();
	WRITE	dataFrame.toJSON().saveAsTextFile(<path to location>,classOf[Compression Codec])

Problem Scenario 1

PLEASE READ THE INTRODUCTION TO THIS SERIES. CLICK ON HOME LINK AND READ THE INTRO BEFORE ATTEMPTING TO SOLVE THE PROBLEMS

Video walk through of the solution to this problem can be found here ***[Click here]***

Click here for the video version of this series. This takes you to the youtube playlist of videos.

Problem 1:

1. Using sqoop, import orders table into hdfs to folders **/user/cloudera/problem1/orders**. File should be loaded as Avro File and use snappy compression
2. Using sqoop, import order_items table into hdfs to folders **/user/cloudera/problem1/order-items**. Files should be loaded as avro file and use snappy compression
3. Using Spark Scala load data at **/user/cloudera/problem1/orders** and **/user/cloudera/problem1/orders-items** items as *dataframes*.
4. **Expected Intermediate Result:** Order_Date , Order_status, total_orders, total_amount. In plain english, please find total orders and total amount per status per day. The result should be sorted by order date in descending, order status in ascending and total amount in descending and total orders in ascending. Aggregation should be done using below methods. However, sorting can be done using a dataframe or RDD. Perform aggregation in each of the following ways
    - a). Just by using Data Frames API - here order_date should be YYYY-MM-DD format
    - b). Using Spark SQL - here order_date should be YYYY-MM-DD format
    - c). By using combineByKey function on RDDS -- No need of formatting order_date or total_amount
5. Store the result as parquet file into hdfs using gzip compression under folder
    - /user/cloudera/problem1/result4a-gzip
    - /user/cloudera/problem1/result4b-gzip
    - /user/cloudera/problem1/result4c-gzip
6. Store the result as parquet file into hdfs using snappy compression under folder
    - /user/cloudera/problem1/result4a-snappy
    - /user/cloudera/problem1/result4b-snappy
    - /user/cloudera/problem1/result4c-snappy
7. Store the result as CSV file into hdfs using No compression under folder
    - /user/cloudera/problem1/result4a-csv
    - /user/cloudera/problem1/result4b-csv
    - /user/cloudera/problem1/result4c-csv
8. create a mysql table named result and load data from **/user/cloudera/problem1/result4a-csv** to mysql table named result

Solution:

Try your best to solve the above scenario without going through the solution below. If you could then use the solution to compare your result. If you could not then I strongly recommend that you go through the concepts again (this time in more depth). Each step below provides a solution to the points mentioned in the Problem Scenario.

Step 1:

```
sqoop import \
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
--username retail_dba \
--password cloudera \
--table orders \
--compress \
--compression-codec org.apache.hadoop.io.compress.SnappyCodec \
--target-dir /user/cloudera/problem1/orders \
--as-avrodatafile;
```

Step 2:

```
sqoop import \
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
--username retail_dba \
--password cloudera \
--table order_items \
--compress \
--compression-codec org.apache.hadoop.io.compress.SnappyCodec \
--target-dir /user/cloudera/problem1/order-items \
--as-avrodatafile;
```

Step 3:

```
import com.databricks.spark.avro._;
  var ordersDF = sqlContext.read.avro("/user/cloudera/problem1/orders");
  var orderItemDF = sqlContext.read.avro("/user/cloudera/problem1/order-items");
```

Step 4:

```
var joinedOrderDataDF = ordersDF
.join(orderItemDF,ordersDF("order_id")===orderItemDF("order_item_order_id"))
```

Step 4a:

```
import org.apache.spark.sql.functions._;



var dataFrameResult =
dataFrameResult.show();



joinedOrderDataDF.
groupBy(to_date(from_unixtime(col("order_date")/1000)).alias("order_formatted_date"),col("order_status")).

agg(round(sum("order_item_subtotal"),2).alias("total_amount"),countDistinct("order_id").alias("total_orders")).

orderBy(col("order_formatted_date").desc,col("order_status"),col("total_amount").desc,col("total_orders"));
```

Step 4b:

```
joinedOrderDataDF.registerTempTable("order_joined");


var sqlResult = sqlContext.sql("select to_date(from_unixtime(cast(order_date/1000 as bigint))) as order_formatted_date, order_status,
cast(sum(order_item_subtotal) as DECIMAL (10,2)) as total_amount, count(distinct(order_id)) as total_orders from order_joined group by
to_date(from_unixtime(cast(order_date/1000 as bigint))), order_status order by order_formatted_date desc,order_status,total_amount desc,
total_orders");


sqlResult.show();
```

Step 4c:

```
var comByKeyResult =
joinedOrderDataDF.
map(x=> ((x(1).toString,x(3).toString),(x(8).toString.toFloat,x(0).toString))).
combineByKey((x:(Float, String))=>(x._1,Set(x._2)),
(x:(Float,Set[String]),y:(Float,String))=>(x._1 + y._1,x._2+y._2),
(x:(Float,Set[String]),y:(Float,Set[String]))=>(x._1+y._1,x._2++y._2)).
map(x=> (x._1._1,x._1._2,x._2._1,x._2._2.size)).
toDF().
orderBy(col("_1").desc,col("_2"),col("_3").desc,col("_4"));

comByKeyResult.show();
```

Step 5:

- sqlContext.setConf("spark.sql.parquet.compression.codec","gzip");
- dataFrameResult.write.parquet("/user/cloudera/problem1/result4a-gzip");

- sqlResult.write.parquet("/user/cloudera/problem1/result4b-gzip");
- comByKeyResult.write.parquet("/user/cloudera/problem1/result4c-gzip");

Step 6:

- sqlContext.setConf("spark.sql.parquet.compression.codec","snappy");
- dataFrameResult.write.parquet("/user/cloudera/problem1/result4a-snappy");
- sqlResult.write.parquet("/user/cloudera/problem1/result4b-snappy");
- comByKeyResult.write.parquet("/user/cloudera/problem1/result4c-snappy");

Step 7:

- dataFrameResult.map(x=> x(0) + "," + x(1) + "," + x(2) + "," + x(3)).saveAsTextFile("/user/cloudera/problem1/result4a-csv")
- sqlResult.map(x=> x(0) + "," + x(1) + "," + x(2) + "," + x(3)).saveAsTextFile("/user/cloudera/problem1/result4b-csv")
- comByKeyResult.map(x=> x(0) + "," + x(1) + "," + x(2) + "," + x(3)).saveAsTextFile("/user/cloudera/problem1/result4c-csv")

Step 8:

a) login to MYSQL using below : mysql -h localhost -u retail_dba -p
(when prompted password use cloudera or any password that you have currently set)

b) create table retail_db.result(order_date varchar(255) not null,order_status varchar(255) not null, total_orders int, total_amount numeric, constraint pk_order_result primary key (order_date,order_status));

c)

```
sqoop export \
--table result \
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
--username retail_dba \
--password cloudera \
--export-dir "/user/cloudera/problem1/result4a-csv" \
--columns "order_date,order_status,total_amount,total_orders"
```

Problem Scenario 2

PLEASE READ THE INTRODUCTION TO THIS SERIES. CLICK ON HOME LINK AND READ THE INTRO BEFORE ATTEMPTING TO SOLVE THE PROBLEMS

Video walk through of the solution to this problem can be found here ***[Click here]***

Click here for the video version of this series. This takes you to the youtube playlist of videos.

Problem 2:

1. Using sqoop copy data available in mysql products table to folder **/user/cloudera/products** on hdfs as text file. columns should be delimited by pipe '|'
2. move all the files from **/user/cloudera/products** folder to **/user/cloudera/problem2/products** folder
3. Change permissions of all the files under **/user/cloudera/problem2/products** such that owner has read,write and execute permissions, group has read and write permissions whereas others have just read and execute permissions
4. read data in **/user/cloudera/problem2/products** and do the following operations using **a)** dataframes api **b)** spark sql **c)** RDDs aggregateByKey method. Your solution should have three sets of steps. Sort the resultant dataset by category id
    - filter such that your RDD\DF has products whose price is lesser than 100 USD
    - on the filtered data set find out the higest value in the product_price column under each category
    - on the filtered data set also find out total products under each category
    - on the filtered data set also find out the average price of the product under each category
    - on the filtered data set also find out the minimum price of the product under each category
5. store the result in avro file using snappy compression under these folders respectively
    - /user/cloudera/problem2/products/result-df
    - /user/cloudera/problem2/products/result-sql
    - /user/cloudera/problem2/products/result-rdd

Solution:

Try your best to solve the above scenario without going through the solution below. If you could then use the solution to compare your result. If you could not then I strongly recommend that you go through the concepts again (this time in more depth). Each step below provides a solution to the points mentioned in the Problem Scenario.

Step 1:

```
sqoop import \
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
--username retail_dba \
--password cloudera \
--table products \
```

```
--as-textfile \
--target-dir /user/cloudera/products \
--fields-terminated-by '|';
```

Step 2:

```
hadoop fs -mkdir /user/cloudera/problem2/
hadoop fs -mkdir /user/cloudera/problem2/products
hadoop fs -mv /user/cloudera/products/* /user/cloudera/problem2/products/
```

Step 3:

```
//Read is 4, Write is 2 and execute is 1.
//ReadWrite,Execute = 4 + 2 + 1 = 7
//Read,Write = 4+2 = 6
//Read ,Execute=4+1=5

hadoop fs -chmod 765 /user/cloudera/problem2/products/*
```

Step 4:

```
scala> var products = sc.textFile("/user/cloudera/products").map(x=> {var d = x.split('|');
(d(0).toInt,d(1).toInt,d(2).toString,d(3).toString,d(4).toFloat,d(5).toString)});


scala>case class Product(productID:Integer, productCatID: Integer, productName: String,
productDesc:String, productPrice:Float, productImage:String);


scala> var productsDF = products.map(x=> Product(x._1,x._2,x._3,x._4,x._5,x._6)).toDF();
```

Step 4-Data Frame Api:

```
scala> import org.apache.spark.sql.functions._
scala> var dataFrameResult = productsDF.filter("productPrice <
100").groupBy(col("productCategory")).agg(max(col("productPrice")).alias("max_price"),countDistin
```

```
ct(col("productID")).alias("tot_products"),round(avg(col("productPrice")),2).alias("avg_price"),min(c
ol("productPrice")).alias("min_price")).orderBy(col("productCategory"));

scala> dataFrameResult.show();
```

Step 4 - Spark SQL:

```
productsDF.registerTempTable("products");
var sqlResult = sqlContext.sql("select product_category_id, max(product_price) as maximum_price,
count(distinct(product_id)) as total_products, cast(avg(product_price) as decimal(10,2)) as
average_price, min(product_price) as minimum_price from products where product_price <100
group by product_category_id order by product_category_id desc");
sqlResult.show();
```

Step 4 - RDD aggregateByKey:

```
var rddResult =
productsDF.map(x=>(x(1).toString.toInt,x(4).toString.toDouble)).aggregateByKey((0.0,0.0,0,9999999
999999.0))((x,y)=>(math.max(x._1,y),x._2+y,x._3+1,math.min(x._4,y)),(x,y)=>(math.max(x._1,y._1),x.
_2+y._2,x._3+y._3,math.min(x._4,y._4))).map(x=>
(x._1,x._2._1,(x._2._2/x._2._3),x._2._3,x._2._4)).sortBy(_._1, false);
rddResult.collect().foreach(println);
```

Step 5:

```
-> import com.databricks.spark.avro._;
-> sqlContext.setConf("spark.sql.avro.compression.codec","snappy")
->dataFrameResult.write.avro("/user/cloudera/problem2/products/result-df");
->sqlResult.write.avro("/user/cloudera/problem2/products/result-sql");
->rddResult.toDF().write.avro("/user/cloudera/problem2/products/result-rdd");;
```

Problem Scenario 3

PLEASE READ THE INTRODUCTION TO THIS SERIES. CLICK ON HOME LINK AND READ THE INTRO BEFORE ATTEMPTING TO SOLVE THE PROBLEMS

Video walk through of this solution is available at ***[Click Here]***

Click here for the video version of this series. This takes you to the youtube playlist of videos.

Problem 3: Perform in the same sequence

1. Import all tables from mysql database into hdfs as avro data files. use compression and the compression codec should be snappy. data warehouse directory should be **retail_stage.db**
2. Create a metastore table that should point to the orders data imported by sqoop job above. Name the table **orders_sqoop**.
3. Write query in hive that shows all orders belonging to a certain day. This day is when the most orders were placed. select data from **orders_sqoop**.
4. query table in impala that shows all orders belonging to a certain day. This day is when the most orders were placed. select data from **order_sqoop.**
5. Now create a table named **retail**.**orders_avro** in hive stored as avro, the table should have same table definition as order_sqoop. Additionally, this new table should be partitioned by the order month i.e -> year-order_month.(example: 2014-01)
6. Load data into orders_avro table from orders_sqoop table.
7. Write query in hive that shows all orders belonging to a certain day. This day is when the most orders were placed. select data from **orders_avro**
8. evolve the avro schema related to **orders_sqoop** table by adding more fields named (order_style String, order_zone Integer)
9. insert two more records into orders_sqoop table.
10. Write query in hive that shows all orders belonging to a certain day. This day is when the most orders were placed. select data from **orders_sqoop**
11. query table in impala that shows all orders belonging to a certain day. This day is when the most orders were placed. select data from **orders_sqoop**

Solution:

Try your best to solve the above scenario without going through the solution below. If you could then use the solution to compare your result. If you could not then I strongly recommend that you go through the concepts again (this time in more depth). Each step below provides a solution to the points mentioned in the Problem Scenario.

Step 1:

```
sqoop import-all-tables \
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
--username retail_dba \
--password cloudera \
--warehouse-dir /user/hive/warehouse/retail_stage.db \
--compress \
```

```
--compression-codec snappy \
--as-avrodatafile
-m 1;
```

Step 2:

```
hadoop fs -get /user/hive/warehouse/retail_stage.db/orders/part-m-00000.avro
avro-tools getschema part-m-00000.avro > orders.avsc
hadoop fs -mkdir /user/hive/schemas
hadoop fs -ls /user/hive/schemas/order
hadoop fs -copyFromLocal orders.avsc /user/hive/schemas/order
```

Launch HIVE using 'hive' command in a separate terminal

Below HIVE command will create a table pointing to the avro data file for orders data

```
create external table orders_sqoop
STORED AS AVRO
LOCATION '/user/hive/warehouse/retail_stage.db/orders'
TBLPROPERTIES ('avro.schema.url'='/user/hive/schemas/order/orders.avsc')
```

Step 3-Run the query in Hive:

Run this query in Hive.

```
select * from orders_sqoop as X where X.order_date in (select inner.order_date from (select
Y.order_date, count(1) as total_orders from orders_sqoop as Y group by Y.order_date order by
total_orders desc, Y.order_date desc limit 1) inner);
```

Step 4-Run the query Impala:

Lanch Impala shell by using command impala-shell

1. Run 'Invalidate metadata'
2. Run below query

```
select * from orders_sqoop as X where X.order_date in (select a.order_date from (select Y.order_date, count(1) as total_orders from orders_sqoop as Y group by Y.order_date order by total_orders desc, Y.order_date desc limit 1) a);
```

Step 5 and 6:

```
create database retail;

create table orders_avro
    > (order_id int,
    > order_date date,
    > order_customer_id int,
    > order_status string)
    > partitioned by (order_month string)
    > STORED AS AVRO;

 insert overwrite table orders_avro partition (order_month)
select order_id, to_date(from_unixtime(cast(order_date/1000 as int))), order_customer_id, order_status, substr(from_unixtime(cast(order_date/1000 as int)),1,7) as order_month from default.orders_sqoop;
```

Step 7 - Query Hive

```
select * from orders_avro as X where X.order_date in (select inner.order_date from (select Y.order_date, count(1) as total_orders from orders_avro as Y group by Y.order_date order by total_orders desc, Y.order_date desc limit 1) inner);
```

Step 8 - Evolve Avro Schema

1. hadoop fs -get /user/hive/schemas/order/orders.avsc
2. gedit orders.avsc

3.

```
{
  "type" : "record",
  "name" : "orders",
  "doc" : "Sqoop import of orders",
```

```
 "fields" : [ {
  "name" : "order_id",
  "type" : [ "null", "int" ],
  "default" : null,
  "columnName" : "order_id",
  "sqlType" : "4"
 }, {
  "name" : "order_date",
  "type" : [ "null", "long" ],
  "default" : null,
  "columnName" : "order_date",
  "sqlType" : "93"
 }, {
  "name" : "order_customer_id",
  "type" : [ "null", "int" ],
  "default" : null,
  "columnName" : "order_customer_id",
  "sqlType" : "4"
 },{
  "name" : "order_style",
  "type" : [ "null", "string" ],
  "default" : null,
  "columnName" : "order_style",
  "sqlType" : "12"
 }, {
  "name" : "order_zone",
  "type" : [ "null", "int" ],
  "default" : null,
  "columnName" : "order_zone",
  "sqlType" : "4"
 }, {
  "name" : "order_status",
  "type" : [ "null", "string" ],
  "default" : null,
  "columnName" : "order_status",
  "sqlType" : "12"
 } ],
 "tableName" : "orders"
}
```

4. hadoop fs -copyFromLocal -f orders.avsc /user/hive/schemas/order/orders.avsc

Step 9 - Insert 2 records from Hive shell

```
insert into table orders_sqoop values (8888888,1374735600000,11567,"xyz",9,"CLOSED");
```

insert into table orders_sqoop values (8888889,1374735600000,11567,"xyz",9,"CLOSED");

Step 10 -Run the query in Hive:

Run this query in Hive.

```
select * from orders_sqoop as X where X.order_date in (select inner.order_date from (select Y.order_date, count(1) as total_orders from orders_sqoop as Y group by Y.order_date order by total_orders desc, Y.order_date desc limit 1) inner);
```

Step 11-Run the query Impala:

Lanch Impala shell by using command impala-shell

1. Run 'Invalidate metadata'
2. Run below query

```
select * from orders_sqoop as X where X.order_date in (select a.order_date from (select Y.order_date, count(1) as total_orders from orders_sqoop as Y group by Y.order_date order by total_orders desc, Y.order_date desc limit 1) a);
```

Problem Scenario 4

PLEASE READ THE INTRODUCTION TO THIS SERIES. CLICK ON HOME LINK AND READ THE INTRO BEFORE ATTEMPTING TO SOLVE THE PROBLEMS

Video walk-through of the solution to this problem can be found here ***[Click here]***

Click here for the video version of this series. This takes you to the youtube playlist of videos.

In this problem, we will focus on conversion between different file formats using spark or hive. This is a very import examination topic. I recommend that you master the data file conversion techniques and understand the limitations. You should have an alternate method of accomplishing a solution to the problem in case your primary method fails for any unknown reason. For example, if saving the result as a text file with snappy compression fails while using spark then you should be able to accomplish the same thing using Hive. In this blog\video I am going to walk you through some scenarios that cover alternative ways of dealing with same problem.

Problem 4:

1. Import orders table from mysql as text file to the destination **/user/cloudera/problem5/text.** Fields should be terminated by a tab character ("\t") character and lines should be terminated by new line character ("\n").
2. Import orders table from mysql into hdfs to the destination **/user/cloudera/problem5/avro**. File should be stored as avro file.
3. Import orders table from mysql into hdfs to folders **/user/cloudera/problem5/parquet**. File should be stored as parquet file.
4. Transform/Convert data-files at **/user/cloudera/problem5/avro** and store the converted file at the following locations and file formats
    - save the data to hdfs using snappy compression as parquet file at **/user/cloudera/problem5/parquet-snappy-compress**
    - save the data to hdfs using gzip compression as text file at **/user/cloudera/problem5/text-gzip-compress**
    - save the data to hdfs using no compression as sequence file at **/user/cloudera/problem5/sequence**
    - save the data to hdfs using snappy compression as text file at **/user/cloudera/problem5/text-snappy-compress**
5. Transform/Convert data-files at **/user/cloudera/problem5/parquet-snappy-compress** and store the converted file at the following locations and file formats
    - save the data to hdfs using no compression as parquet file at **/user/cloudera/problem5/parquet-no-compress**
    - save the data to hdfs using snappy compression as avro file at **/user/cloudera/problem5/avro-snappy**
6. Transform/Convert data-files at **/user/cloudera/problem5/avro-snappy** and store the converted file at the following locations and file formats
    - save the data to hdfs using no compression as json file at **/user/cloudera/problem5/json-no-compress**
    - save the data to hdfs using gzip compression as json file at **/user/cloudera/problem5/json-gzip**
7. Transform/Convert data-files at **/user/cloudera/problem5/json-gzip** and store the converted file at the following locations and file formats
    - save the data to as comma separated text using gzip compression at **/user/cloudera/problem5/csv-gzip**
8. Using spark access data at **/user/cloudera/problem5/sequence** and stored it back to hdfs using no compression as ORC file to HDFS to destination **/user/cloudera/problem5/orc**

Solution:

Try your best to solve the above scenario without going through the solution below. If you could then use the solution to compare your result. If you could not then I strongly recommend that you go through the concepts again (this time in more depth). Each step below provides a solution to the points mentioned in the Problem Scenario.

Step 1:

```
sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" --password cloudera --username retail_dba --table orders --as-textfile --fields-terminated-by '\t' --target-dir /user/cloudera/problem5/text -m 1
```

Step 2:

```
sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" --password cloudera --username retail_dba --table orders --as-avrodatafile--target-dir /user/cloudera/problem5/avro -m 1
```

Step 3:

```
sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" --password cloudera --username retail_dba --table orders --as-parquetfile --target-dir /user/cloudera/problem5/parquet -m 1
```

Step 4:

```
var dataFile = sqlContext.read.avro("/user/cloudera/problem5/avro");

sqlContext.setConf("spark.sql.parquet.compression.codec","snappy");

dataFile.repartition(1).write.parquet("/user/cloudera/problem5/parquet-snappy-compress");

dataFile.map(x=> x(0)+"\t"+x(1)+"\t"+x(2)+"\t"+x(3)).saveAsTextFile("/user/cloudera/problem5/text-gzip-compress",classOf[org.apache.hadoop.io.compress.GzipCodec]);

dataFile.map(x=>
(x(0).toString,x(0)+"\t"+x(1)+"\t"+x(2)+"\t"+x(3))).saveAsSequenceFile("/user/cloudera/problem5/sequence");
```

Below may fail in some cloudera VMS. If the spark command fails use the sqoop command to accomplish the problem. Remember you need to get out to spark shell to run the sqoop command.

```
dataFile.map(x=> x(0)+"\t"+x(1)+"\t"+x(2)+"\t"+x(3)).saveAsTextFile("/user/cloudera/problem5/text-snappy-compress",classOf[org.apache.hadoop.io.compress.SnappyCodec]);
```

```
sqoop import --table orders --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" --username retail_dba --password cloudera --as-textfile -m1 --target-dir user/cloudera/problem5/text-snappy-compress --compress --compression-codec org.apache.hadoop.io.compress.SnappyCodec
```

Step 5:

```
var parquetDataFile = sqlContext.read.parquet("/user/cloudera/problem5/parquet-snappy-compress")

sqlContext.setConf("spark.sql.parquet.compression.codec","uncompressed");

parquetDataFile.write.parquet("/user/cloudera/problem5/parquet-no-compress");
```

```
sqlContext.setConf("spark.sql.avro.compression.codec","snappy");



parquetDataFile.write.avro("/user/cloudera/problem5/avro-snappy");
```

Step 6:

```
var avroData = sqlContext.read.avro("/user/cloudera/problem5/avro-snappy");

avroData.toJSON.saveAsTextFile("/user/cloudera/problem5/json-no-compress");

avroData.toJSON.saveAsTextFile("/user/cloudera/problem5/json-
gzip",classOf[org.apache.hadoop.io.GzipCodec]);
```

Step 7 :

```
var jsonData = sqlContext.read.json("/user/cloudera/problem5/json-gzip");

jsonData.map(x=>x(0)+","+x(1)+","+x(2)+","+x(3)).saveAsTextFile("/user/cloudera/problem5/csv-
gzip",classOf[org.apache.hadoop.io.compress.GzipCodec])
```

Step 8:

```
//To read the sequence file you need to understand the sequence getter for the key and value class
to //be used while loading the sequence file as a spark RDD.
//In a new terminal Get the Sequence file to local file system
hadoop fs -get /user/cloudera/problem5/sequence/part-00000

//read the first 300 characters to understand the two classes to be used.

cut -c-300 part-00000



//In spark shell do below

var seqData =
sc.sequenceFile("/user/cloudera/problem5/sequence/",classOf[org.apache.hadoop.io.Text],classOf[
org.apache.hadoop.io.Text]);

seqData.map(x=>{var d = x._2.toString.split("\t");
(d(0),d(1),d(2),d(3))}).toDF().write.orc("/user/cloudera/problem5/orc");
```

Contribution from **Raphael L. Nascimento:**

You can use below method as well for setting the compression codec.

```
sqlContext.sql("SET spark.sql.parquet.compression.codec=snappy")
```

Problem Scenario 5 [SQOOP]

CCA 175 Hadoop and Spark Developer Exam Preparation - Problem Scenario 5

PLEASE READ THE INTRODUCTION TO THIS SERIES. CLICK ON HOME LINK AND READ THE INTRO BEFORE ATTEMPTING TO SOLVE THE PROBLEMS

Video walkthrough of this problem is available at ***[PART 1 CLICK HERE]*** AND ***[PART 2 CLICK HERE]***

Click here for the video version of this series. This takes you to the youtube playlist of videos.

Sqoop is one of the important topics for the exam. Based on generally reported exam pattern from anonymous internet bloggers, you can expect 2 out of 10 questions on this topic related to Data Ingest and Data Export using Sqoop. Hence, 20% of the exam score can be obtained just by practicing simple Sqoop concepts. Sqoop can be mastered easily (i.e in a few hours) at the skill level that CCA 175 exam is expecting you to demonstrate. I created this problem focusing on Sqoop alone, if you are able to perform this exercise on your own or at worst using just the sqoop user guide then there is a very very high chance that you can score the 20% easily.

Pre-Work: Please perform these steps before solving the problem

1. Login to MySQL using below commands on a fresh terminal window
   mysql -u retail_dba -p
   Password = cloudera
2. Create a replica product table and name it products_replica
   create table products_replica as select * from products
3. Add primary key to the newly created table
   alter table products_replica add primary key (product_id);
4. Add two more columns
   alter table products_replica add column (product_grade int, product_sentiment varchar(100))
5. Run below two update statements to modify the data
   update products_replica set product_grade = 1 where product_price > 500;
   update products_replica set product_sentiment = 'WEAK' where product_price between 300 and 500;

Problem 5: Above steps are important so please complete them successfully before attempting to solve the problem

1. Using sqoop, import products_replica table from MYSQL into hdfs such that fields are separated by a '|' and lines are separated by '\n'. Null values are represented as -1 for numbers and "NOT-AVAILABLE" for strings. Only records with product id greater than or equal to 1 and less than or equal to 1000 should be imported and use 3 mappers for importing. The destination file should be stored as a text file to directory **/user/cloudera/problem5/products-text**.

2. Using sqoop, import products_replica table from MYSQL into hdfs such that fields are separated by a '*' and lines are separated by '\n'. Null values are represented as -1000 for numbers and "NA" for strings. Only records with product id less than or equal to 1111 should be imported and use 2 mappers for importing. The destination file should be stored as a text file to directory **/user/cloudera/problem5/products-text-part1**.
3. Using sqoop, import products_replica table from MYSQL into hdfs such that fields are separated by a '*' and lines are separated by '\n'. Null values are represented as -1000 for numbers and "NA" for strings. Only records with product id greater than 1111 should be imported and use 5 mappers for importing. The destination file should be stored as a text file to directory **/user/cloudera/problem5/products-text-part2.**
4. Using sqoop merge data available in **/user/cloudera/problem5/products-text-part1** and **/user/cloudera/problem5/products-text-part2** to produce a new set of files in **/user/cloudera/problem5/products-text-both-parts**
5. Using sqoop do the following. Read the entire steps before you create the sqoop job.
    - create a sqoop job Import Products_replica table as text file to directory **/user/cloudera/problem5/products-incremental**. Import all the records.
    - insert three more records to Products_replica from mysql
    - run the sqoop job again so that only newly added records can be pulled from mysql
    - insert 2 more records to Products_replica from mysql
    - run the sqoop job again so that only newly added records can be pulled from mysql
    - Validate to make sure the records have not be duplicated in **HDFS**
6. Using sqoop do the following. Read the entire steps before you create the sqoop job.
    - create a hive table in database named **problem5** using below command
    - create table **products_hive** (product_id int, product_category_id int, product_name string, product_description string, product_price float, product_imaage string,product_grade int, product_sentiment string);
    - create a sqoop job Import Products_replica table as hive table to database named **problem5**. name the table as **products_hive**.
    - insert three more records to Products_replica from mysql
    - run the sqoop job again so that only newly added records can be pulled from mysql
    - insert 2 more records to Products_replica from mysql
    - run the sqoop job again so that only newly added records can be pulled from mysql
    - Validate to make sure the records have not been duplicated in **Hive** table
7. Using sqoop do the following. .
    - insert 2 more records into **products_hive** table using hive.
    - create table in mysql using below command
    - create table products_external (product_id int(11) primary Key, product_grade int(11), product_category_id int(11), product_name varchar(100), product_description varchar(100), product_price float, product_impage varchar(500), product_sentiment varchar(100));
    - export data from products_hive (hive) table to (mysql) products_external table.
    - insert 2 more records to Products_hive table from hive
    - export data from products_hive table to products_external table.
    - Validate to make sure the records have not be duplicated in **mysql** table

Solution:

Try your best to solve the above scenario without going through the solution below. If you could then use the solution to compare your result. If you could not then I strongly recommend that you

go through the concepts again (this time in more depth). Each step below provides a solution to the points mentioned in the Problem Scenario.

Step 1:

```
sqoop import \
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
--username retail_dba \
--password cloudera \
--table products_replica \
--target-dir /user/cloudera/problem5/products-text \
--fields-terminated-by '|' \
--lines-terminated-by '\n' \
--null-non-string -1 \
--null-string "NOT-AVAILABLE" \
-m 3 \
--where "product_id between 1 and 1000" \
--outdir /home/cloudera/sqoop1 \
--boundary-query "select min(product_id), max(product_id) from products_replica where
product_id between 1 and 1000";
```

Step 2:

```
sqoop import \
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
--username retail_dba \
--password cloudera \
--table products_replica \
--target-dir  /user/cloudera/problem5/products-text-part1  \
--fields-terminated-by '*' \
--lines-terminated-by '\n' \
--null-non-string -1000 \
--null-string "NA" \
-m 2 \
--where "product_id <= 1111 " \
--outdir /home/cloudera/sqoop2 \
--boundary-query "select min(product_id), max(product_id) from products_replica where
product_id <= 1111";
```

Step 3:

```
sqoop import \
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
--username retail_dba \
--password cloudera \
--table products_replica \
--target-dir  /user/cloudera/problem5/products-text-part2  \
--fields-terminated-by '*' \
--lines-terminated-by '\n' \
--null-non-string -1000 \
--null-string "NA" \
-m 5 \
--where "product_id > 1111 " \
--outdir /home/cloudera/sqoop3 \
--boundary-query "select min(product_id), max(product_id) from products_replica where
product_id > 1111"
```

Step 4:

```
sqoop merge \
--class-name products_replica \
--jar-file mp/sqoop-cloudera/compile/66b4f23796be7625138f2171a7331cd3/products_replica.jar \
--new-data /user/cloudera/problem5/products-text-part2 \
--onto /user/cloudera/problem5/products-text-part1 \
--target-dir /user/cloudera/problem5/products-text-both-parts \
--merge-key product_id;
```

Step 5:

On terminal -

```
sqoop job --create first_sqoop_job \
-- import \
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
--username "retail_dba" \
--password "cloudera" \
--table products_replica \
--target-dir /user/cloudera/problem5/products-incremental \
--check-column product_id \
--incremental append \
--last-value 0;
```

```
sqoop job --exec first_sqoop_job
```

On MySQL command line -

```
mysql> insert into products_replica values (1346,2,'something 1','something 2',300.00,'not avaialble',3,'STRONG');

mysql> insert into products_replica values (1347,5,'something 787','something 2',356.00,'not avaialble',3,'STRONG');
```

On terminal -

```
sqoop job --exec first_sqoop_job
```

On MYSQL Command Line

```
insert into products_replica values (1376,4,'something 1376','something 2',1.00,'not avaialble',3,'WEAK');

insert into products_replica values (1365,4,'something 1376','something 2',10.00,'not avaialble',null,'NOT APPLICABLE');
```

On terminal -

```
sqoop job --exec first_sqoop_job
```

Step 6:

On Terminal window-

```
sqoop job \
--create hive_sqoop_job \
-- import \
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
--username "retail_dba" \
```

```
--password "cloudera" \
--table products_replica  \
--check-column product_id \
--incremental append \
--last-value 0 \
--hive-import \
--hive-table products_hive \
--hive-database problem5;
```

On Hive window:

```
create database problem5;

use problem5;

create table products_hive  (product_id int, product_category_id int, product_name string,
product_description string, product_price float, product_imaage string,product_grade
int,  product_sentiment string);
```

On Terminal window

```
sqoop job --exec hive_sqoop_job
```

On MySQL window

```
insert into products_replica values (1378,4,'something 1376','something 2',10.00,'not
avaialble',null,'NOT APPLICABLE');

insert into products_replica values (1379,4,'something 1376','something 2',10.00,'not
avaialble',null,'NOT APPLICABLE');
```

On Terminal Window

```
sqoop job --exec hive_sqoop_job
```

On Hive Window

```
select * from products_hive
```

Step 7:

On Hive Window

```
use problem5;

insert into table products_hive values (1380,4,'something 1380','something 2',8.00,'not avaialble',3,'NOT APPLICABLE');

insert into table products_hive values (1381,4,'something 1380','something 2',8.00,'not avaialble',3,'NOT APPLICABLE');
```

On MYSQL window

```
create table products_external  (product_id int(11) primary Key, product_grade int(11), product_category_id int(11), product_name varchar(100), product_description varchar(100), product_price float, product_impage varchar(500), product_sentiment varchar(100));
```

On Terminal

```
sqoop export \
--username "retail_dba" \
--password "cloudera" \
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
 --export-dir /user/hive/warehouse/problem5.db/products_hive/ \
--input-fields-terminated-by '\001' \
--input-null-non-string "null" \
```

```
--input-null-string "null" \

--update-mode allowinsert \

--update-key product_id \

--columns
"product_id,product_category_id,product_name,product_description,product_price,product_impag
e,product_grade,product_sentiment" --table products_external;
```

On Hive Window

```
insert into table products_hive values (1382,4,'something 1380','something 2',8.00,'not
avaialble',3,'NOT APPLICABLE');


insert into table products_hive values (1383,4,'something 1380','something 2',8.00,'not
avaialble',3,'NOT APPLICABLE');
```

On Terminal Window:

```
sqoop export \
--username "retail_dba" \
--password "cloudera" \
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
--export-dir /user/hive/warehouse/problem5.db/products_hive/ \
--input-fields-terminated-by '\001' \
--input-null-non-string "null" \
--input-null-string "null" \
--update-mode allowinsert \
--update-key product_id \
--columns
"product_id,product_category_id,product_name,product_description,product_price,product_impag
e,product_grade,product_sentiment" --table products_external;
```

To Validate

On Hive

```
select count(*) from problem5.products_hive;
```

on MySQL

```
select count(*) from products_replica;
```

Problem Scenario 6 [Data Analysis]

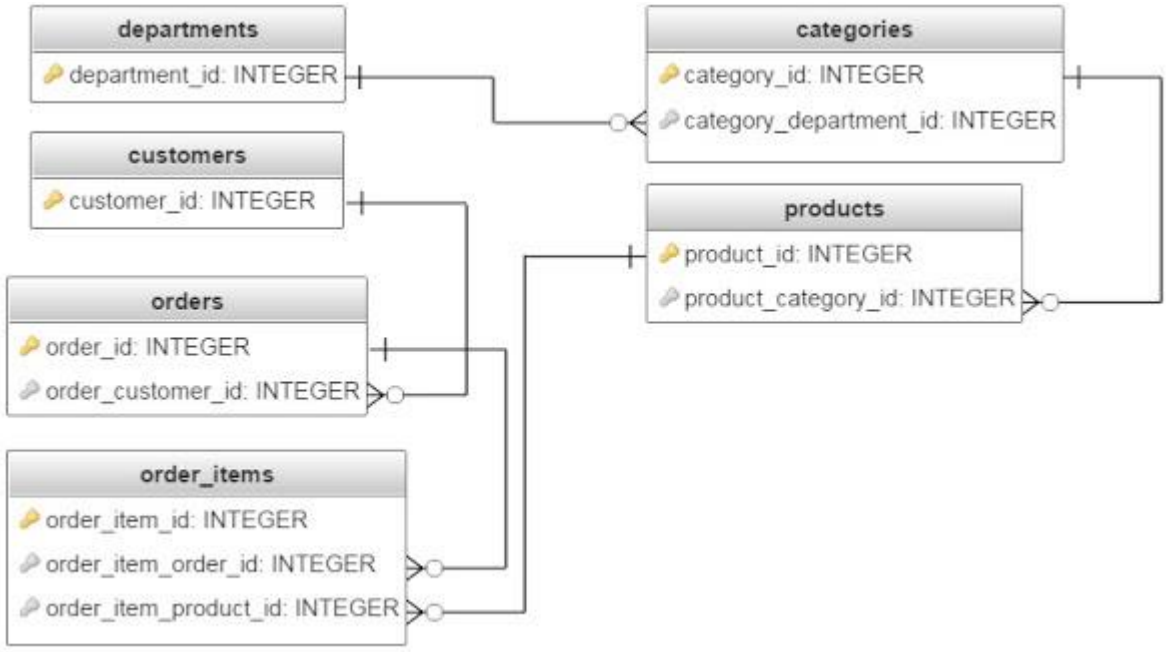
PLEASE READ THE INTRODUCTION TO THIS SERIES. CLICK ON HOME LINK AND READ THE INTRO BEFORE ATTEMPTING TO SOLVE THE PROBLEMS

Video walkthrough of this problem is available at ***[CLICK HERE]*** AND

Click here for the video version of this series. This takes you to the youtube playlist of videos.

This problem helps you strengthen and validate skills related to ***data analysis*** objective of the certification exam.

Data model in mysql on cloudera VM looks like this. [*Note: only primary and foreign keys are included in the relational schema diagram shown below*]



Problem 6: Provide two solutions for steps 2 to 7

- Using HIVE QL over Hive Context
- Using Spark SQL over Spark SQL Context or by using RDDs

1. create a hive meta store database named **problem6** and import all tables from mysql retail_db database into hive meta store.
2. On spark shell use data available on meta store as source and perform step 3,4,5 and 6. *[this proves your ability to use meta store as a source]*
3. Rank products within department by price and order by department ascending and rank descending *[this proves you can produce ranked and sorted data on joined data sets]*

4. find top 10 customers with most unique product purchases. if more than one customer has the same number of product purchases then the customer with the lowest customer_id will take precedence *[this proves you can produce aggregate statistics on joined datasets]*
5. On dataset from step 3, apply filter such that only products less than 100 are extracted *[this proves you can use subqueries and also filter data]*
6. On dataset from step 4, extract details of products purchased by top 10 customers which are priced at less than 100 USD per unit *[this proves you can use subqueries and also filter data]*
7. Store the result of 5 and 6 in new meta store tables within hive. *[this proves your ability to use metastore as a sink]*

Solution:

Try your best to solve the above scenario without going through the solution below. If you could then use the solution to compare your result. If you could not then I strongly recommend that you go through the concepts again (this time in more depth). Each step below provides a solution to the points mentioned in the Problem Scenario. Please go through the video for an indepth explanation of the solution.

NOTE: The same solution can be implemented using Spark SQL Context. Just replace Hive Context object with SQL Context object below. Rest of the solution remains the same. i.e same concept of querying, using temp table and storing the result back to hive.

Step 1:

```
sqoop import-all-tables --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" --username retail_dba --password cloudera --warehouse-dir /user/hive/warehouse/problem6.db --hive-import --hive-database problem6 --create-hive-table --as-textfile;
```

Step 2:

```
var hc = new org.apache.spark.sql.hive.HiveContext(sc);
```

Step 3:

```
var hiveResult =
hc.sql("select  d.department_id,  p.product_id,  p.product_name,  p.product_price,  rank() over (partition by d.department_id order by p.product_price) as product_price_rank,  dense_rank() over (partition by d.department_id order by p.product_price) as product_dense_price_rank  from products p  inner join categories c on c.category_id = p.product_category_id  inner join departments d on c.category_department_id = d.department_id  order by d.department_id, product_price_rank desc, product_dense_price_rank ");
```

Step 4:

```
var hiveResult2 = hc.sql("select c.customer_id, c.customer_fname, count(distinct(oi.order_item_product_id)) unique_products  from customers c  inner join orders o on o.order_customer_id = c.customer_id  inner join order_items oi on o.order_id = oi.order_item_order_id  group by c.customer_id, c.customer_fname  order by unique_products
```

```
desc, c.customer_id   limit 10
")
```

Step 5:

```
hiveResult.registerTempTable("product_rank_result_temp");

hc.sql("select * from product_rank_result_temp where product_price < 100").show();
```

Step 6:

```
 var topCustomers = hc.sql("select c.customer_id, c.customer_fname,
count(distinct(oi.order_item_product_id)) unique_products  from customers c   inner join orders o
on o.order_customer_id = c.customer_id  inner join order_items oi on o.order_id =
oi.order_item_order_id  group by c.customer_id, c.customer_fname  order by unique_products
desc, c.customer_id   limit 10  ");



topCustomers.registerTempTable("top_cust");



var topProducts = hc.sql("select distinct p.* from products p inner join order_items oi on
oi.order_item_product_id = p.product_id inner join orders o on o.order_id = oi.order_item_order_id
inner join top_cust tc on o.order_customer_id = tc.customer_id where p.product_price < 100");
```

Step 7:

```
hc.sql("create table problem6.product_rank_result as select * from product_rank_result_temp
where product_price < 100");



hc.sql("create table problem 6.top_products as select distinct p.* from products p inner join
order_items oi on oi.order_item_product_id = p.product_id inner join orders o on o.order_id =
oi.order_item_order_id inner join top_cust tc on o.order_customer_id = tc.customer_id where
p.product_price < 100");
```

Problem Scenario 7 [FLUME]

CCA 175 Hadoop and Spark Developer Exam Preparation - Problem Scenario 7

PLEASE READ THE INTRODUCTION TO THIS SERIES. CLICK ON HOME LINK AND READ THE INTRO BEFORE ATTEMPTING TO SOLVE THE PROBLEMS

Video walkthrough of this problem is available at [*CLICK HERE*]

Click here for the video version of this series. This takes you to the youtube playlist of videos.

This question focusses on validating your ***flume*** skills. You can either learn flume by following the video accompanied with this post or learn flume elsewhere and then solve this problem while using the video as a reference. This video serves both as tutorial and walkthrough of how to leverage flume for data ingestion.

Note: While this post only provides specifics related to solving the problem, the video provides an introduction, explanation and more importantly application of flume knowledge.

Problem 7:

1. This step comprises of three substeps. Please perform tasks under each subset completely
    - using sqoop pull data from MYSQL **orders** table into **/user/cloudera/problem7/prework** as **AVRO** data file using only one mapper
    - Pull the file from **\user\cloudera\problem7\prework** into a local folder named **flume-avro**
    - create a flume agent configuration such that it has an avro source at localhost and port number 11112, a jdbc channel and an hdfs file sink at **/user/cloudera/problem7/sink**
    - Use the following command to run an avro client **flume-ng avro-client -H localhost -p 11112 -F <<Provide your avro file path here>>**
2. The CDH comes prepackaged with a log generating job. **start_logs, stop_logs** and **tail_logs**. Using these as an aid and provide a solution to below problem. The generated logs can be found at path **/opt/gen_logs/logs/access.log**
    - run start_logs
    - write a flume configuration such that the logs generated by start_logs are dumped into HDFS at location **/user/cloudera/problem7/step2.** The channel should be non-durable and hence fastest in nature. The channel should be able to hold a maximum of **1000** messages and should commit after every **200** messages.
    - Run the agent.
    - confirm if logs are getting dumped to hdfs.
    - run stop_logs.

Solution:

Step 1:

Pull orders data from order sqoop table to **\user\cloudera\problem7\prework**

```
sqoop import --table orders --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" --username retail_dba --password cloudera -m 1 --target-dir /user/cloudera/problem7/prework --as-avrodatafile
```

Get the file from HDFS to local

```
mkdir flume-avro;

cd flume-avro;

hadoop fs -get /user/cloudera/problem7/prework/* .

gedit f.config
```

Create a flume-config file in problem7 folder named **f.config**

```
#Agent Name = step1

# Name the source, channel and sink
step1.sources = avro-source
step1.channels = jdbc-channel
step1.sinks = file-sink

# Source configuration
step1.sources.avro-source.type = avro
step1.sources.avro-source.port = 11112
step1.sources.avro-source.bind = localhost


# Describe the sink
step1.sinks.file-sink.type = hdfs
step1.sinks.file-sink.hdfs.path = /user/cloudera/problem7/sink
step1.sinks.file-sink.hdfs.fileType = DataStream
step1.sinks.file-sink.hdfs.fileSuffix = .avro
step1.sinks.file-sink.serializer = avro_event
step1.sinks.file-sink.serializer.compressionCodec=snappy
```

```
# Describe the type of channel --  Use memory channel if jdbc channel does not work
step1.channels.jdbc-channel.type = jdbc

# Bind the source and sink to the channel
step1.sources.avro-source.channels = jdbc-channel
step1.sinks.file-sink.channel = jdbc-channel
```

Run the flume agent

```
flume-ng agent --name step1 --conf . --conf-file f.config
```

Run the flume Avro client

```
flume-ng avro-client -H localhost -p 11112 -F <<Provide your avro file path here>>
```

Step 2:

```
mkdir flume-logs
cd flume-logs
```

create flume configuration file

```
# Name the components on this agent
a1.sources = r1
a1.sinks = k1
a1.channels = c1

# Describe/configure the source
a1.sources.r1.type = exec
a1.sources.r1.command = tail -F /opt/gen_logs/logs/access.log


# Describe the sink
a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path = /user/cloudera/problem7/step2
a1.sinks.k1.hdfs.fileSuffix = .log
a1.sinks.k1.hdfs.writeFormat = Text
a1.sinks.k1.hdfs.fileType = DataStream

# Use a channel which buffers events in memory
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 200
```

```
# Bind the source and sink to the channel
a1.sources.r1.channels = c1

a1.sinks.k1.channel = c1
```

create hdfs sink directory

```
hadoop fs -mkdir /user/cloudera/problem7/sink
```

Run the flume-agent

```
flume-ng agent --name a1 --conf . --conf-file f.config
```