Technical assessment for a Big Data developer

The purpose of the assessment is to test your ability to:

- Import data to Hive
- Use Spark and Scala efficiently
- Integrate with other systems using Kafka

What is expected?

The application you will be writing stores and transforms data about residential mortgages based securities. You will import data to HIVE, do simple transformation on it and save it to a different database. Then you will publish a message to a Kafka consumer to notify that the process is done.

Step 1:

Write an application that accepts and file name of a ZIP file that includes a DAT file with Securities information. The application will extract this file and import its contents into a Hive table called security_raw. The structure of the table can be found here:

http://embs.com/public/html/PostProcessedFileFmt.htm#Sec

Please use the attached file: "GNM_SEC.ZIP-2017-05-08_23-58-17.ZIP" to test your application.

You can use any programming language to write this application.

<<<SOLUTION>>>

1. Download zip file into ~/Downloads and extract file using:

```
#!/bin/bash
if [ "$#" -ne 1 ]; then
  echo "Please supply zip filename"
  exit 1
fi
FILE=$1
echo $FILE
unzip ~/Downloads/$FILE
```

2. Import the content into Hive table
    - create a file (sequrity-raw.hsql) with following contents:

      create external table if not exists security_raw(SecId integer, SecMnem char(20), ProductSupType char(6), Product char(12), CUSIP char(9), Agency char(3), Prefix char(2), PrefixId integer, IssAmt double, CollType char(4), IssDt date, MatDt date, OrigWac double, OrigWam integer, OrigWala integer, RetireDt date, IssueId integer, PriceId integer, TBAEligCode char(3), SsnCode char(3), InsSrc char(4), InsDt char(17), UpdSrc char(4), UpdDt char(17))

      row format delimited fields terminated by '|';

      load data local inpath "/path-where-zip-was-extracted/GNM_SEC.dat" into table security_raw;

    - Run Hive command to create the table
        - **hive -f sequrity-raw.hsql**

Step 2:

Write a Spark application using Scala that will read security_raw and create a new table called security_normalised. This table should be in ORC format and it will be identical to security_raw except of the following changes:

1. Drop the columns: PrefixId, InsSrc, UpdSrc
2. Create a new column called: NormalisationDate -> this will be the timestamp of the insertion time of this record to your new table (UTC).
3. Create a new boolean column called: IsNew. If the InsDt column of the record is in the last 12 months then the value will be true, otherwise false.

<<<SOLUTION>>>

scala> spark.sql("desc security_raw").collect().foreach(println)

[secid,int,null]

[secmnem,string,null]

[productsuptype,string,null]

[product,string,null]

[cusip,string,null]

[agency,string,null]

[prefix,string,null]

[prefixid,int,null]

[issamt,double,null]

[colltype,string,null]

[issdt,date,null]

[matdt,date,null]

[origwac,double,null]

[origwam,int,null]

[origwala,int,null]

[retiredt,date,null]

[issueid,int,null]

[priceid,int,null]

[tbaeligcode,string,null]

[ssncode,string,null]

[inssrc,string,null]

[insdt,string,null]

[updsrc,string,null]

[upddt,string,null]

scala> spark.sql("create table security_normalised stored as ORC as select * from security_raw")

res2: org.apache.spark.sql.DataFrame = []

//not allowed for table stored as ORC

spark.sql("alter table security_normalised replace columns (secid int,secmnem string,productsuptype string,product string,cusip string,agency string,prefix string,issamt double,colltype string,issdt date,matdt date,origwac double,origwam int,origwala int,retiredt date,issueid int,priceid int,tbaeligcode string,ssncode string,insdt string,upddt string)")

scala> spark.sql("drop table security_normalised")

res14: org.apache.spark.sql.DataFrame = []

scala> spark.sql("create table security_normalised stored as ORC as select secid,secmnem,productsuptype,product,cusip,agency,prefix,issamt,colltype,issdt,matdt,origwac,origwam,origwala,retiredt,issueid,priceid,tbaeligcode,ssncode,insdt,upddt from security_raw")

res15: org.apache.spark.sql.DataFrame = []

scala> spark.sql("select count(*) from security_normalised").collect().foreach(println)

[5367]

scala> spark.sql("drop table security_normalised")

scala> spark.sql("create table security_normalised stored as ORC TBLPROPERTIES ('transactional'='true') as select secid,secmnem,productsuptype,product,cusip,agency,prefix,issamt,colltype,issdt,matdt,origwac,origwam,origwala,retiredt,issueid,priceid,tbaeligcode,ssncode,insdt,upddt,**cast(unix_timestamp() as timestamp) as NormalisationDate, case when Insdt >= from_unixtime(unix_timestamp(now() - interval 365 days), 'yyyyMMdd HH:mm:ss') then cast(1 as boolean) else cast(0 as boolean) end as IsNew** from security_raw")

scala> spark.sql("select count(*) from security_normalised").collect().foreach(println)

[5367]

scala> spark.sql("select secid,secmnem,productsuptype,product,InsDt,NormalisationDate,IsNew from security_normalised where Insdt >= from_unixtime(unix_timestamp(now() - interval 365 days), 'yyyyMMdd HH:mm:ss') limit 5").collect().foreach(println)

[3589825,GNM784189C ,POOL ,GNMII30C ,20160921 19:17:00,2017-08-31 23:40:20.0,true]

[3589826,GNM784190C ,POOL ,GNMII30C ,20160921 19:17:00,2017-08-31 23:40:20.0,true]

[3589306,GNM784191C ,POOL ,GNMII30C ,20160914 19:15:00,2017-08-31 23:40:20.0,true]

[3589920,GNM784193X ,POOL ,GNM30 ,20160923 19:16:00,2017-08-31 23:40:20.0,true]

[3590059,GNM784194M ,POOL ,GNMII30M ,20160930 19:24:00,2017-08-31 23:40:20.0,true]

Step 3:

Create a Kafka topic called normalisation_completed, then amend your Scala application so it sends a message when it finished creating security_normalised.

<<<SOLUTION>>>

```
// send message "Success" to "normalisation_completed" topic
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="normalisation_completed"
val record = new ProducerRecord(topic, "key","Success " + java.time.LocalDateTime.now)
producer.send(record)
producer.close()
```

Kafka

//Start the server

zookeeper-server-start.sh ${KFAKA_HOME}/config/zookeeper.properties &

kafka-server-start.sh ${KFAKA_HOME}/config/server.properties &

//Create “normalisation_completed” topic

kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic normalisation_completed

//Start a consumer

kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic normalisation_completed --from-beginning

<<< Spark program - SparkEmbsSecurity.scala >>>>>>

```
import java.io.File

import java.util.Properties

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

import org.apache.spark.sql.SparkSession

object SparkEmbsSecurity {

  def main (args: Array[String]) {

    val spark = SparkSession

      .builder()

      .appName("SparkEmbsSecurity")

      .config("spark.master", "local")

     .enableHiveSupport()

     .getOrCreate()

    import spark.implicits._

    import spark.sql

    // using data from security_raw table, create a new table

    spark.sql("create table if not exists security_normalised stored as ORC TBLPROPERTIES ('transactional'='true') as select secid,secmnem,productsuptype,product,cusip,agency,prefix,issamt,colltype,issdt,matdt,origwac,origwam,origwala,retiredt,issueid,priceid,tbaeligcode,ssncode,insdt,upddt,cast(unix_timestamp() as timestamp) as NormalisationDate, case when Insdt >= from_unixtime(unix_timestamp(now() - interval 365 days), 'yyyyMMdd HH:mm:ss') then cast(1 as boolean) else cast(0 as boolean) end  as IsNew from security_raw")

    spark.sql("select secid,secmnem,productsuptype,product,InsDt,NormalisationDate,IsNew from security_normalised where Insdt >= from_unixtime(unix_timestamp(now() - interval 365 days), 'yyyyMMdd HH:mm:ss') limit 5").collect().foreach(println)

    // if table stored as ORC then alter table is not allowed or you cannot update the table

   spark.stop()

    // send message "Success" to "normalisation_completed" topic

    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="normalisation_completed"
    val record = new ProducerRecord(topic, "key","Success " + java.time.LocalDateTime.now)
    producer.send(record)
    producer.close()
  }
}
```

<<< build.sbt >>>>>>

```
name := "embs data processing"
version := "1.0"
val kafkaVersion = "0.11.0.0"
scalaVersion := "2.11.8"
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.2.0"
libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.2.0"
libraryDependencies += "org.apache.spark" %% "spark-hive" % "2.2.0"
libraryDependencies += "org.apache.kafka" % "kafka-clients" % kafkaVersion
resolvers += Resolver.mavenLocal
```

<<< Make jar file >>>>>>

$ sbt package

<<< Run **SparkEmbsSecurity** >>>>>>

```
$ spark-submit --class SparkEmbsSecurity --jars
${LOCATION_OF_KAFKA_CLIENTS_JAR_FILE}/kafka-clients-0.11.0.0.jar
${LOCATION_JAR}/embs-data-processing_2.11-1.0.jar
```

-------- http://embs.com/public/html/PostProcessedFileFmt.htm - tables --------

create external table Sec(SecId integer, SecMnem char(20), ProductSupType char(6), Product char(12), CUSIP char(9), Agency char(3), Prefix char(2), PrefixId integer, IssAmt double, CollType char(4), IssDt date, MatDt date, OrigWac double, OrigWam integer, OrigWala integer, RetireDt date, IssueId integer, PriceId integer, TBAEligCode char(3), SsnCode char(3), InsSrc char(4), InsDt char(17), UpdSrc char(4), UpdDt char(17))

create external table SecCurr(CUSIP char(9), SecMnem char(20), SecId integer, Coupon float, CouponType char(3), CpnEffDt date, CpnPublType char(4), IssuerId integer, GnmIssNr integer, Wac double, WacSrc char(2), Wam integer, WamSrc char(2), PublWam integer, Wala integer, WalaSrc char(2), PublWala integer, WtdAvgEffDt date, ProductionYear char(4), WaolTerm integer, WaolSize double, Fctr double, FctrDt date, PublFctr double, PublFctrDt date, CurtailFctr double, Fctr1 double, Fctr3 double, Fctr6 double, Fctr12 double, Fctr24 double, Cpr1 float, Cpr3 float, Cpr6

float, Cpr12 float, Cpr24 float, CprLife float, Psa1 float, Psa3 float, Psa6 float, Psa12 float, Psa24 float, PsaLife float, UpdatedLongestMatDt date, WAOLTV integer, WAOCS integer, WACLSize float, CurrNumLoans integer, CLEffDt date, UpdSrc char(4), UpdDt char(17))

create external table Hist(SecId integer, CUSIP char(9), EffDt date, FctrJan double, FctrSrcJan char(4), CprmJan float, WacJan float, WacSrcJan char(4), WamJan integer, WamSrcJan char(4), WalaJan integer, WalaSrcJan char(4))

create external table Issuer(IssuerId integer, IssuerName varchar(60), GnmIssNr integer, IssuerSt char(2), IssuerAddr varchar(60), IssuerCity varchar(40), IssuerZip char(9), Contact1Name varchar(40), Contact1Tel varchar(13), Contact2Name varchar(40), Contact2Tel varchar(13), IssuerFullString varchar(200))

create external table Prefix(PrefixId integer, Agency char(3), Prefix char(2), PoolNrMin char(6), PoolNrMax char(6), GnmaPlan char(1), Product char(12), MktTkr char(6), Description varchar(150), AmortPer integer, OrigTerm integer, DaysToFirstPay integer, ServFeeDflt float, CouponType char(3), PymtType char(5), PropertyType char(5), LoanType char(5), CollType char(4), PrdTBAEligCode char(3), TotPymtIncrPer integer, FirstPymtIncrPer integer, PymtIncrPer integer, PymtIncrRt float, PrePymtLockoutPer integer, CalcCprFlag char(1), CalcPsaFlag char(1), CalcPYFlag char(1), VariableTermFlag char(1))

create external table GeoLnDst(RecType char(1), Agency char(3), PoolNr char(7), Cusip char(9), IssueId integer, EffDt date, Value char(3), OrigYr char(4), RPB float, PctRPB float, Loans integer)

create external table LoanDist(RecType char(1), Agency char(3), PoolNumber char(7), Cusip char(9), IssueId integer, EffDt date, Value1 char(6), RPB float, PctRPB float, Loans integer)

create external table LoanDist2(RecType char(1), Agency char(3), PoolNr char(7), Cusip char(9), IssueId integer, EffDt date, Value1 char(6), Value2 char(4), Data1 int, RPB float, PctRPB float, Loans integer)

create external table Servicer(RecType char(1), Agency char(3), PoolNr char(7), Cusip char(9), IssueId integer, EffDt date, Servicer char(40), Value2 char(4), RPB float, PctRPB float, Loans integer, WAC float, NoteRateHigh float, NoteRateLow float, WALA float, LoanAgeHigh float, LoanAgeLow float, WARM float, RemMatHigh float, RemMatLow float)

create external table Quartile(IssueId integer, EffDt date, IndOrigUpd char(1), Field char(6), WtdAvg float, Q1Min float, Q1Max float, Q2Min (FHLMC only) float, Q2Max float, Q3Min (FHLMC only) float, Q3Max float, Q4Min (FHLMC only) float, Q4Max float)

create external table AdjRate(SecId int, EffDt date, Margin real, CpnAdjDt date, OrigIntRate float, ProspIntRate float, IndexRefDt date, PaymtAdjDt date, IndexDescr varchar(80), HARMType char(5), Lookback int, LookbackPaymt int, MtgAdjPer int, PerRateCap real, ConvertibleFlag char(1), ConvertBeginDt date, ConvertEndDt date,

ErrorFlag char(1), LifeCeil real, LifeFloor real, NextCpnAdjDt date, OrigWtdAvgMtgMargin real, OrigMtgMarginHi real, OrigMtgMarginLo real, OrigWtdAvgMtgLifeCeil real, OrigMtgLifeCeilHi real, OrigMtgLifeCeilLo real, OrigWtdAvgMtgLifeFloor real, OrigMtgLifeFloorHi real, OrigMtgLifeFloorLo real, WtdAvgCompLifeFloor real, OrigMtgCpnHi real, OrigMtgCpnLo real, OrigRemMtgMatHi int, OrigRemMtgMatLo int, WtdAvgMonthsToAdj real, OrigCompMarginDiff real, OrigPctUPB int, SubType char(4), TransferType char(1), Structure char(1), DeferredIntAllowedFlag char(1), OrigNumLoans int, RateAdjFreq int, PaymtChgFreq int, AmortRecastFreq int, PaymtCap real, AccrRtRoundMethod char(2), MinIndexMove real, RateDiffFlag char(1), PublPassThroughRate float, EstAccrualRate float, WtdAvgLoanMargin real, WtdAvgNegAmortLimit real, WtdAvgLoanLifeCap real, WtdAvgLoanLifeFloor real, CurrDeferredInt real, RtrnDeferredInt real, FirstPaymtDt date, NegAmortFctr float, InitCapUpPct float, InitCapDownPct float, ServicingBPFlag char(1), AssumabilityCode char(1), PpyProtMtgFlag char(1), IndexId int, IndexGrp char(5), ResetPerInit int, ResetPerSubseq int, CapStruct char(20), SubseqPerCap float, LifeCap float, InterestOnlyPerFlag char(1), UpdSrc char(4), UpdDt date)

create external table ARMReset(SecId int, AdjDt date, RPB real, WtdAvgLoanCoupon real, LoanCouponHi real, LoanCouponLo real, WtdAvgLoanMargin real, LoanMarginHi real, LoanMarginLo real, WtdAvgLoanLifeCap real, LoanLifeCapHi real, LoanLifeCapLo real, WtdAvgLoanLifeFloor real, LoanLifeFloorHi real, LoanLifeFloorLo real, NumLoans int, EffDt date, UpdSrc char(4), UpdDt date)

create external table Aggr(AggrId integer, AggrType integer, AggrLvl integer, AggrRecType char(3), Agency char(3), CollType char(4), CouponType char(5), Product char(12), Prefix char(3), Coupon real, ProdnYr char(14), DrillDown varchar(40), EffDt date YYYYMMDD, SecMnem char(20), IssAmt float, SecCountAll integer, RPB float, SecCount integer, AvgWac real, AvgWam real, AvgWalaCurr real, AvgOCS real, AvgOLTV real, AvgOSize real, NewIss0 real, Cpr1,Cpr2...Cpr48 real, NewIss1,NewIss2...NewIss48 real, Rpb1,Rpb2...Rpb48 real, NewIssWac1,NewIssWac2... NewIssWac48 real, NewIssWam1,NewIssWam2... NewIssWam48 smallint, NewIssWala1,NewIssWala2... NewIssWala48 smallint, AvgCpr3 real, AvgCpr6 real, AvgCpr12 real, AvgCpr24 real, AvgCpr36 real, ProdnYrMin char(4), ProdnYrMax char(4), UpdDt smalldatetime, GnmaUpd char(1))