PySpark

Cloudera Folks

Task 1 ---
open pyspark shell --- (type pyspark and enter)

```
value='zeyobron'
sc.parallelize([value]).foreach(print)
```

Lab Folks --- 1pm

```
value='zeyobron'
sc.parallelize([value]).collect()
```

Task 2 --- (Optional)
Cloudera Folks
open terminal

```
cd
echo 1,sai>data
hadoop fs -put data /user/cloudera/
```

pyspark and go inside

```
spark.read.format("csv").load("/user/cloudera/data").show()
```

Scala Spark

```
val file1 = sc.textFile("file:///home/cloudera/revdata/file1.txt")
file1.foreach(println)
```

Py Spark

```
file1 = sc.textFile("file:///home/cloudera/revdata/file1.txt")
file1.foreach(print)
```

Scala Spark

```
val gymdata = file1.filter( x => x.contains("Gymnastics"))
gymdata.foreach(println)
```

Py Spark

```
--------------------------------
gymdata = file1.filter( lambda x :  'Gymnastics' in x )
gymdata.foreach(print)
```

```
===============================
```

Scala Spark

```
case class
schema(txnno:String,txndate:String,custno:String,amount:String,category:String,p
roduct:String,city:String,state:String,spendby:String)
val mapsplit=gymdata.map(x => x.split(","))
val schemardd = mapsplit.map( x =>
schema(x(0),x(1),x(2),x(3),x(4),x(5),x(6),x(7),x(8)))
val schemafilter = schemardd.filter( x => x.product.contains("Gymnastics"))
schema.foreach(println)
-------------------------------------
```

Py Spark

```
from collections import namedtuple
schema=
namedtuple("schema",["txnno","txndate","custno","amount","category","product","ci
ty","state","spendby"])

mapsplit = gymdata.map( lambda x : x.split(","))
schemardd = mapsplit.map(lambda x :
schema(x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7],x[8]))
schemafilter = schemardd.filter( lambda x : 'Gymnastics' in x.product)
schemafilter.foreach(print)
```

```
===============================
```

Scala Spark

```
val schemadf = schemafilter.toDF()
schemadf.show()
```

Py Spark

```
schemadf = schemafilter.toDF()
schemadf.show()
==============================
```

Scala Spark

```
val file2= sc.textFile("file:///home/cloudera/revdata/file2.txt")
val rowmapsplit = file2.map( x => x.split(","))
import org.apache.spark.sql.Row
val rowrdd = rowmapsplit.map( x => Row(x(0),x(1),x(2),x(3),x(4),x(5),x(6),x(7),x(8)))
rowrdd.foreach(println)
```

Py Spark

```
file2= sc.textFile("file:///home/cloudera/revdata/file2.txt")
rowmapsplit = file2.map(lambda x : x.split(","))
from pyspark.sql import Row
rowrdd = rowmapsplit.map(lambda x :
Row(x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7],x[8]))
rowrdd.foreach(print)
```

Scala Spark---------

```
import org.apache.spark.sql.types._
val structschema = StructType(Array(
   StructField("txnno",StringType,true),
   StructField("txndate",StringType,true),
   StructField("custno",StringType,true),
   StructField("amount", StringType, true),
   StructField("category", StringType, true),
   StructField("product", StringType, true),
   StructField("city", StringType, true),
   StructField("state", StringType, true),
   StructField("spendby", StringType, true)
 ))
val rowdf = spark.createDataFrame(rowrdd,structschema)
rowdf.show()
```

Py Spark

```
from pyspark.sql.types import *
structschema = StructType([ \
   StructField("txnno",StringType(),True), \
   StructField("txndate",StringType(),True), \
   StructField("custno",StringType(),True), \
   StructField("amount", StringType(), True), \
   StructField("category", StringType(), True), \
   StructField("product", StringType(), True), \
   StructField("city", StringType(), True), \
   StructField("state", StringType(), True), \
   StructField("spendby", StringType(), True) \
  ])

rowdf = spark.createDataFrame(rowrdd,structschema)
rowdf.show()
```

==============================

Py Spark

```
csvdf =
spark.read.format("csv").option("header","true").load("file:///home/cloudera/revd
ata/file3.txt")

jsondf =
spark.read.format("json").load("file:///home/cloudera/revdata/file4.json")
parquetdf = spark.read.load("file:///home/cloudera/revdata/file5.parquet")

xmldf =
spark.read.format("xml").option("rowtag","txndata").load("file:///home/cloudera/r
evdata/file6")

collist =
["txnno","txndate","custno","amount","category","product","city","state","spendby"
]

schemadf1 = schemadf.select(*collist)
```

```
rowdf1 = rowdf.select(*collist)
csvdf1 = csvdf.select(*collist)
jsondf1= jsondf.select(*collist)
parquetdf1 = parquetdf.select(*collist)
xmldf1 = xmldf.select(*collist)

uniondf =
schemadf1.union(rowdf1).union(csvdf1).union(jsondf1).union(parquetdf1).union(xmldf
1)
```
==============================

Scala Spark

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

val resdf =uniondf.withColumn("txndate",expr("split(txndate,'-
')[2]")).withColumnRenamed("txndate","year").withColumn("status",expr("case when
spendby='cash' then 1 else 0 end")).filter(col("txnno")>50000)
```

Py Spark

```
from pyspark.sql.functions import *
resdf =uniondf.withColumn("txndate",expr("split(txndate,'-
')[2]")).withColumnRenamed("txndate","year").withColumn("status",expr("case when
spendby='cash' then 1 else 0 end")).filter(col("txnno")>50000)
```

==============================

Scala Spark

```
val aggdf =
resdf.groupBy("category").agg(sum("amount").cast(IntegerType()).alias("total"))
```

Py Spark
```
aggdf =
resdf.groupBy("category").agg(sum("amount").cast(IntegerType()).alias("total"))
```
==============================

Scala Spark

```
uniondf.write.format("parquet").partitionBy("category").mode("overwrite").save("/u
ser/cloudera/revdirectory")
```

Py Spark

```
uniondf.write.format("parquet").partitionBy("category").mode("overwrite").save("/u
ser/cloudera/revdirectory")
```

==============================

Scala Spark

```
val cust =
spark.read.format("csv").option("header","true").load("file:///home/cloudera/revd
ata/cust.csv")

val prod =
spark.read.format("csv").option("header","true").load("file:///home/cloudera/revd
ata/prod.csv")

val inner = cust.join(prod,Seq("id"),"inner")
val left = cust.join(prod,Seq("id"),"left")
val right = cust.join(prod,Seq("id"),"right")
val full = cust.join(prod,Seq("id"),"full")
val anti = cust.join(prod,Seq("id"),"left_anti")
```

Py Spark

```
cust =
spark.read.format("csv").option("header","true").load("file:///home/cloudera/revd
ata/cust.csv")
prod =
spark.read.format("csv").option("header","true").load("file:///home/cloudera/revd
ata/prod.csv")

inner = cust.join(prod,["id"],"inner")
left = cust.join(prod,["id"],"left")
right = cust.join(prod,["id"],"right")
```

```
full = cust.join(prod,["id"],"full")
anti = cust.join(prod,["id"],"left_anti")

==============================

Py Spark

actor =
spark.read.format("json").option("multiline","true").load("file:///home/cloudera/re
vdata/actorsj.json")
actor.printSchema()
flattendf = actor.withColumn("Actors",explode(col("Actors")))
flattendf.printSchema()
finalflatten=
flattendf.select("Actors.Birthdate","Actors.BornAt","Actors.age","Actors.hasChild
ren","Actors.hasGreyHair","Actors.name","Actors.photo","Actors.picture.*","Actor
s.weight","Actors.wife","country","version")
finalflatten.printSchema()
finalflatten.show()
==============================

Py Spark
=============
Python 3
=============

import urllib.request
import ssl
context = ssl.create_default_context()
context.check_hostname = False
context.verify_mode = ssl.CERT_NONE
url = "https://randomuser.me/api/0.8/?results=500"
response = urllib.request.urlopen(url, context=context).read().decode('utf-8')
urlstring = response
print(urlstring)
```

=============
Python 2.7
=============

```
import urllib2
import ssl

# Disable SSL certificate verification by creating a custom SSL context
context = ssl.create_default_context()
context.check_hostname = False
context.verify_mode = ssl.CERT_NONE

# Make the HTTP request
url = "https://randomuser.me/api/0.8/?results=500"
response = urllib2.urlopen(url, context=context)

# Check if the request was successful
if response.getcode() == 200:
    content = response.read()
    urlstring = content
else:
    print("Failed to fetch data. Status code:", response.getcode())
    urlstring = None

# Do whatever you need with the 'urlstring' variable here
```

======================================

Pyspark Project Code
================================
https://randomuser.me/api/0.8/?results=500

```
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

import urllib.request
import ssl
context = ssl.create_default_context()
context.check_hostname = False
context.verify_mode = ssl.CERT_NONE
url = "https://randomuser.me/api/0.8/?results=500"
response = urllib.request.urlopen(url, context=context).read().decode('utf-8')
urlstring = response
print(urlstring)

conf =
SparkConf().setAppName("first").set("spark.driver.allowMultipleContexts","true")
sc = SparkContext(conf)

spark=SparkSession.builder.getOrCreate()

rdd = sc.parallelize([urlstring])
df = spark.read.json(rdd)
df.show()

arrayflatten= df.withColumn("results",expr("explode(results)"))

finalflatten = arrayflatten.select(
"nationality",
"results.user.cell",
"results.user.username",
"results.user.dob",
"results.user.email",
"results.user.gender",
"results.user.location.city",
"results.user.location.state",
"results.user.location.street",
"results.user.location.zip",
"results.user.md5",
"results.user.name.first",
"results.user.name.last",
"results.user.name.title",
"results.user.password",
"results.user.phone",
"results.user.picture.large",
"results.user.picture.medium",
"results.user.picture.thumbnail",
"results.user.registered",
"results.user.salt",
"results.user.sha1",
"results.user.sha256",
"seed",
"version"
)

finalflatten.show()

avrodf = spark.read.format("parquet")
              .load("file:///home/cloudera/revdata/projectsample.parquet")
avrodf.show
avrodf.printSchema()

numdf = finalflatten.withColumn("username",regexp_replace(col("username"),  "([0-
9])", ""))
numdf.show()

joindf = avrodf.join(numdf,["username"],"left")
joindf.show()

availablecustomerinapi=joindf.filter("nationality is not null")
availablecustomerinapi.show()
```

```
notavailablecustomerinapi=joindf.filter("nationality is  null")
notavailablecustomerinapi.show()

availablecustomerinapi.write.format("parquet").mode("append").save("/user/clouder
a/available")

notavailablecustomerinapi.write.format("parquet").mode("append").save("/user/clou
dera/notavailable")
```

Spark Streaming + Kafka Integration Guide

==
https://spark.apache.org/docs/2.2.0/streaming-kafka-0-10-integration.html

Task 1 ------

Remove Tmp folder
Start zookeeper (zookeeper commands)
Start kafka (kafka commands)
Create a topic sparktk (or use existing topic if you created any)
Open eclipse/Intellij and add spark jars
Add kafk spark streaming jars to the project
Use below template and start the stream in eclipse
Start pushing to kafka using producer console

**Change the package name and object if its different

Code
==========

```
package pack
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming._
import org.apache.spark.sql._
import org.apache.spark.sql.functions
import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe
import org.apache.spark.sql.functions._

object obj {
        def main(args:Array[String]):Unit={
                println("Streaming started")
                val conf = new
SparkConf().setAppName("ES").setMaster("local[*]").set("spark.driver.allowMultipl
eContexts","true")
                val sc = new SparkContext(conf)
                sc.setLogLevel("Error")
                val spark = SparkSession
                .builder()
                .getOrCreate()
                import spark.implicits._
                val ssc = new StreamingContext(conf,Seconds(2))
                val topics = Array("sparktk")
                val kafkaParams = Map[String, Object](
                        "bootstrap.servers" -> "localhost:9092",
                        "key.deserializer" -> classOf[StringDeserializer],
                        "value.deserializer" -> classOf[StringDeserializer],
                        "group.id" -> "zeyogroupid",
                        "auto.offset.reset" -> "earliest"
                        )
        val stream = KafkaUtils.createDirectStream[String, String](
                                ssc,
                                PreferConsistent,
                                Subscribe[String, String](topics, kafkaParams)
                                )
                val stream1=stream.map( x =>  x.value)
                stream1.print
                /*stream1.foreachRDD(x=>

                if(!x.isEmpty())
                {
```

```
                val df = x.toDF("value").withColumn("timstamp", current_timestamp)
                                df.show(false)
                }
                                )
*/
                ssc.start()
                ssc.awaitTermination()
        }
}
```

==

Mothers and Father Scenario Working Code

```
====================================================
package pack
import org.apache.spark._
import org.apache.spark.sql._
import org.apache.spark.sql.functions._

object obj {
        def main(args: Array[String]): Unit = {
        val conf = new SparkConf().setAppName("first").setMaster("local[*]")
                val sc = new SparkContext(conf)
                sc.setLogLevel("Error")

                val spark = SparkSession.builder().getOrCreate()
                import spark.implicits._

                val r  = spark.read.format("csv")
                                .option("header","true")
                                .load("file:///C:/data/r.csv")
                                r.show()

                val p  = spark.read.format("csv")
                                .option("header","true")
                                .load("file:///C:/data/p.csv")
                                p.show()

val leftjoin = r.join(p,r("cid")===p("id"),"left").select("cid","pid","name")
                        leftjoin.show()

                val antijoin = p.join(r,r("cid")===p("id"),"left_anti")
                antijoin.show()
```

```
                              val mothers =
antijoin.filter(col("gender")==="M").withColumnRenamed("name","Mothers")
                              mothers.show()

                              val fathers =
antijoin.filter(col("gender")==="F").withColumnRenamed("name","Fathers")
                              fathers.show()

                              val mothersjoin =
leftjoin.join(mothers,leftjoin("pid")===mothers("id"),"left")
                               .select("pid","name","Mothers")
                              mothersjoin.show()

                              val fathersjoin =
mothersjoin.join(fathers,mothersjoin("pid")===fathers("id"),"left")
                              fathersjoin.show()

          val finaldf = fathersjoin.select("name","Mothers","Fathers")
                              finaldf.show()
          }
}
======================
```

Kafka Whole Code
==
```
package pack

import org.apache.spark._
import org.apache.spark._
import org.apache.spark.sql._
import org.apache.spark.streaming._
import org.apache.spark.sql._
import org.apache.spark.sql.functions
import org.apache.kafka.clients.consumer._
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe


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

val conf = new SparkConf().setAppName("first").setMaster("local[*]")
                .set("spark.driver.allowMultipleContexts","true")

                        val sc = new SparkContext(conf)
                        sc.setLogLevel("ERROR")
                        val spark = SparkSession
                                .builder()
                                .getOrCreate()
                        import spark.implicits._
                val ssc = new StreamingContext(conf,Seconds(1))
                val topics = Array("modelcheck")

val kafkaParams = Map[String, Object]("bootstrap.servers" -> "localhost:9092"
                                ,"key.deserializer" -> classOf[StringDeserializer],
                                "value.deserializer" -> classOf[StringDeserializer],
                                "group.id" -> "earlyid",
                                "auto.offset.reset" -> "earliest")

val stream = KafkaUtils.createDirectStream[String,
String](ssc,PreferConsistent,Subscribe[String, String](topics, kafkaParams))

val streamdata=stream.map(record => (record.value))
                streamdata.print()
                ssc.start()
                ssc.awaitTermination()
        }

}
```

=========================
Spark Kinesis Code

```
package pack

import org.apache.spark._
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming.kinesis.KinesisInputDStream
import org.apache.spark.streaming.{Seconds, StreamingContext}
import
com.amazonaws.services.kinesis.clientlibrary.lib.worker.InitialPositionInStream
import org.apache.spark.streaming.Duration
import org.apache.spark._
import org.apache.spark.sql._
import com.amazonaws.protocol.StructuredPojo
import org.apache.spark.sql.functions._
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming._
import
com.amazonaws.services.kinesis.clientlibrary.lib.worker.KinesisClientLibConfiguratio
n
import com.amazonaws.services.dynamodbv2.model.BillingMode
import com.amazonaws.services.cloudwatch.AmazonCloudWatch
import org.apache.hadoop.fs.s3a.S3AFileSystem
import com.fasterxml.jackson.dataformat.cbor.CBORFactory
import com.fasterxml.jackson.core.TSFBuilder
import org.apache.spark.streaming.kinesis.KinesisUtils

object obj {

  def b2s(a: Array[Byte]): String = new String(a)
        def main(args:Array[String]):Unit={
```

```
val conf = new SparkConf().setAppName("first").setMaster("local[*]")
                .set("spark.driver.allowMultipleContexts","true")
        .set("AWS_ACCESS_KEY","AKIAT5PUAWQ7FI7G5YCH")
.set("AWS_SECRET_KEY","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")
.set("AWS_CBOR_DISABLE","true")

        val sc = new SparkContext(conf)
        sc.setLogLevel("ERROR")
        val spark = SparkSession
                .builder()
                .getOrCreate()

        import spark.implicits._
        val ssc = new StreamingContext(conf,Seconds(2))
        val stream= KinesisUtils.createStream(ssc,
                "<ANY UNIQUE GROUP ID>", // groupId
                "<YOUR MQ NAME>",   // YOUR_MQ_NAME
                "https://kinesis.ap-south-1.amazonaws.com",
                "ap-south-1",
        InitialPositionInStream.TRIM_HORIZON, Seconds(2),
                StorageLevel.MEMORY_AND_DISK_SER_2)

        val finalstream=stream.map(x=>b2s(x))
        finalstream.print()

        ssc.start()
        ssc.awaitTermination()

        }

}
```

==============================
spark hbase integration
==============================

```
spark-shell --conf
"spark.driver.extraClassPath=/home/zeyobronstudent2845/hbasejars/*"


import org.apache.spark.sql.{SQLContext, _}
import org.apache.spark.sql.execution.datasources.hbase._
import org.apache.spark.{SparkConf, SparkContext}
import spark.implicits._

def catalog = s"""{
    |"table":{"namespace":"default", "name":"37htab"},
    |"rowkey":"rowkey",
    |"columns":{
    |"srow":{"cf":"rowkey", "col":"rowkey", "type":"string"},
    |"sid":{"cf":"zcf", "col":"id", "type":"string"},
    |"sname":{"cf":"zcf", "col":"name", "type":"string"},
    |"scountry":{"cf":"zcf", "col":"country", "type":"string"}
    |}
|}""".stripMargin

val df = spark.read.options(Map(HBaseTableCatalog.tableCatalog-
>catalog)).format("org.apache.spark.sql.execution.datasources.hbase").load()
df.show()
```

===

```
=====================
*Spark Cassandra Integration*
=====================
package pack

import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.sql._
import org.apache.spark.sql.functions._

object obj1 {
   def main(args:Array[String]):Unit={

   println("spark cassandra integration")
   val conf = new
SparkConf().setAppName("ES").setMaster("local[*]").set("spark.driver.allowMultipl
eContexts","true")
        val sc = new SparkContext(conf)
        sc.setLogLevel("Error")
        val spark = SparkSession
                        .builder()
                        .getOrCreate()
        import spark.implicits._

        val df = spark.read
        .format("org.apache.spark.sql.cassandra")
        .option("spark.cassandra.connection.host","localhost")
        .option("spark.cassandra.connection.port","9042")
        .option("keyspace","b36")
        .option("table","ztab")
        .load()

        df.show()
```

```
    /*          val finaldf = df.filter(col("id")>1)
                finaldf.show()

        finaldf.write.format("org.apache.spark.sql.cassandra")
                    .option("spark.cassandra.connection.host","localhost")
                    .option("spark.cassandra.connection.port","9042")
                    .option("keyspace","b36")
                    .option("table","zout")
                    .save()*/
  }
}
```

============================

Last spark dataframe streaming

=============================

```
package pack
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.sql._
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._

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

val conf = new SparkConf().setAppName("first").setMaster("local[*]")

		val sc = new SparkContext(conf)
		sc.setLogLevel("ERROR")
		val spark= SparkSession.builder.getOrCreate()
		import spark.implicits._

		val schema = StructType(Array(
			StructField("name", StringType, true)));
		val df = spark
		.readStream.format("csv")
		.schema(schema).load("file:///D:/sin/data")

		// df ----- unbounded input table
		val finaldf = df.withColumn("tdate", current_date)

		// finaldf ---- resultant table

	finaldf.writeStream.format("console")
	 .option("checkpointLocation","file:///D:/ch2ekpoint")
			.start()
			.awaitTermination()
	}
}
```

```
=========================================
```

Kafka ReadStream and WriteStream to console

```
=========================================
package pack
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.sql._
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._

object obj {
	def main(args:Array[String]):Unit={
		System.setProperty("hadoop.home.dir", "C:\\hadoop")

	val conf = new SparkConf().setAppName("first").setMaster("local[*]")

	val sc = new SparkContext(conf)
	sc.setLogLevel("ERROR")

	val spark= SparkSession.builder.getOrCreate()
	import spark.implicits._

	val df = spark
		.readStream
		.format("kafka")
		.option("kafka.bootstrap.servers","localhost:9092")
		.option("subscribe","sstk")
		.load()

	val finaldf = df.withColumn("value",expr("cast(value as string)"))

	finaldf.writeStream
		.format("console")
		.option("checkpointLocation","file:///C:/structdatakafka")
		.start()
		.awaitTermination()
	}

}
```

======================
Kafka ReadStream and WriteStream - Write to kafka
======================
Create a kafka topic sstkout and open consumer Console for it
Open producer console for sstk. Run below Code

```
package pack

import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.sql._
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._

object obj {
        def main(args:Array[String]):Unit={
                System.setProperty("hadoop.home.dir", "C:\\hadoop")
        val conf = new SparkConf().setAppName("first").setMaster("local[*]")
                val sc = new SparkContext(conf)
                sc.setLogLevel("ERROR")

                val spark= SparkSession.builder.getOrCreate()
                import spark.implicits._

                val df = spark
                        .readStream
                        .format("kafka")
                        .option("kafka.bootstrap.servers","localhost:9092")
                        .option("subscribe","sstk")
                        .load()
                        .select("value")
                         .withColumn("value",expr("concat(value,',sai')"))
                        .writeStream
                        .format("kafka")
```

```
                    .option("kafka.bootstrap.servers","localhost:9092")
                    .option("topic","sstkout")
                    .option("checkpointLocation","file:///C:/structdatakafka")
                    .start()
                    .awaitTermination()

        }

}

================================================
============================================
Spark dataframe streaming cassandra
============================================

package pack
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.sql._
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.execution.streaming.sources.ForeachBatchSink

object obj {
        def main(args:Array[String]):Unit={
                System.setProperty("hadoop.home.dir", "C:\\hadoop")
        val conf = new SparkConf().setAppName("first").setMaster("local[*]")
                val sc = new SparkContext(conf)
                sc.setLogLevel("ERROR")

                val spark= SparkSession.builder.getOrCreate()
                import spark.implicits._

                val df = spark
                        .readStream
                        .format("kafka")
                        .option("kafka.bootstrap.servers","localhost:9092")
                        .option("subscribe","sstk")
                        .load()
                        .select("value")
                        .withColumn("value",expr("concat(value,',sai')"))
                        .writeStream
                        .foreachBatch  {
                                        (df:DataFrame, id:Long) =>
```

```
                    df.write.format("org.apache.spark.cassandra")
                                .option("spark.cassandra.connection.host","localhost")
                                .option("spark.cassandra.connection.port","9042")
                                .option("keyspace","zeyok")
                                 .option("table","zeyotk")
                                 .save()
                             }
                   .option("checkpointLocation","file:///C:/structdatakafka")
                   .start()
                   .awaitTermination()
         }
}
```

==