**UNIVERSITY OF INFORMATION TECHNOLOGY**

**FACULTY OF INFORMATION SYSTEM**

**GROUP MEMBERS**

* **HOÀNG MINH KHIÊM**
* **NGUYỄN HOÀNG LONG**
* **TRẦN ANH TUẤN**

**PROJECT REPORT**

**SPARK STREAMING**

**HO CHI MINH City, 2019**

**UNIVERSITY OF INFORMATION TECHNOLOGY**

**FACULTY OF INFORMATION SYSTEM**

**GROUP MEMBERS**

* **HOÀNG MINH KHIÊM - 16520588**
* **NGUYỄN HOÀNG LONG - 16520688**
* **TRẦN ANH TUẤN - 16521386**

**PROJECT REPORT**

**SPARK STREAMING**

**LECTURER: Nguyen Thanh Binh**

**HO CHI MINH City, 2019**

Contents

[**1.** **What is Spark?** 4](#_Toc24390798)

[**a.** **Concept:** 4](#_Toc24390799)

[**b.** **Spark Components**: 5](#_Toc24390800)

[**2.** **RDD:** 6](#_Toc24390801)

[**3.** **Spark Context:** 6](#_Toc24390802)

[**5.** **RDD Actions:** 7](#_Toc24390803)

[**6.** **Lazy evaluation:** 8](#_Toc24390804)

[**7.** **Why Spark Streaming:** 8](#_Toc24390805)

[**8.** **Spark Streaming high level overview:** 8](#_Toc24390806)

[**9.** **DStream (Discretized Stream):** 9](#_Toc24390807)

[**10.** **Windowing:** 10](#_Toc24390808)

[**11.** **Spark Streaming Fault-Tolerent:** 11](#_Toc24390809)

[**12.** **How to Install Apache Spark with Scala:** 11](#_Toc24390810)

[**13.** **Learning Scala through sc worksheet(.sc):** 13](#_Toc24390811)

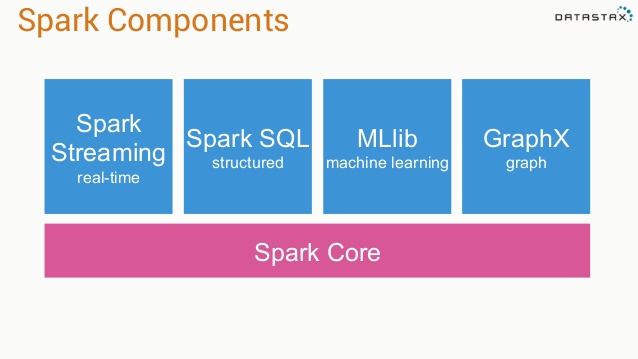
[**14.** **Trying Some Spark Streaming Project with Scala:** 13](#_Toc24390812)

[**15.** **References:** 24](#_Toc24390813)

1. **What is Spark?**
2. **Concept:**

* Spark is a fast and general engine for large-scale data processing.
* Using Spark reduce is a lot easier than using Hadoop map reduce
* Spark contains:
* **Driver program**:
* Instructs spark how to process, transform, output the data.
* **Spark-context**:
* Communicate with your cluster manager if you are using a cluster (Spark, YARN, Mesos).
* Created by Driver program
* Spark maybe a next big thing that replace MapReduce in Hadoop.
* **Cluster Manager:** responsible for controlling the processing of your data across entire cluster.
* **Horizontal Scalable**: Be able to add more and more server and process more and more data.
* It’s 100x times faster than MapReduce in Memory and 10x times in Disk.
* Use **DAG** (Direct acyclic graph) to optimize workflow – Nothing Actually happen until you tell spark to go and do it.  Spark go off and find the best optimal way to approach the problem as a whole.
* Many Big Company use Spark.
* Can be integrated with Java, Python, Scala.

1. **Spark Components**:



* Everything is built on top of **Spark core** which contains all the guts of creating, transforming performing action on RDD.
* **Spark Streaming**: Help us deal with real time constant stream of data that come in forever, we can transform them as they come in.
* **Spark SQL:** Create SQL view of our data out of an RDD.
* **Mllib:** Machine Learning.
* **GraphX:** Graph, Chart.

1. **RDD**

* Stand for: **Resilient Distributed Dataset**
* **Resilient** – Can be distributed and run on a cluster that is actually resilient to failure, if 1 node goes down, RDD can still survive.
* **Distributed** – distribute the processing of RDD. RDD is basically a giant key value data storage.
* **Dataset** – It’s a dataset.

1. **Spark Context:**

* Responsible for making RDD resilient and distributed
* Whenever we start with any spark program, we’ve got to create a spark context first.
* A Spark context object is what creates our rdd objects. It’s like the mastermind behind the whole script
* In **Spark Streaming**, you create a streaming context – Just a special spark context that can deal with real time data.

**Example:**

val ssc = new StreamingContext(“local[\*]”, “LogAlarmer”, Seconds(1))

**Local** = RUN ON THIS PC

**The star \***: spin up a process for every core that I have available on my PC

**LogAlarmer**: name

Process 1 second as a time.

* Creating RDD from data that’s in memory:

**Example:**

Nums = parallelize([1,2,3,4])

* Creating RDD from Text file:

**Example:**

Sc.textFile(“Path”)

* Can Also create from: JDBC, Cassandra, Hbase, Json.

1. **Transforming RDD:**

* Map 1-1
* Flatmap – Take an RDD contains X number of lines and create RDDs that has Y lines (Nothing or 1-N or 1-N)
* Filter
* Distinct
* Sample
* Union, intersection, subtract, cartesian

**Map() example:**

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

Val result = input.map(x => x\*x)

This result in (1, 4, 9, 16).

1. **RDD Actions:**

- Collect

* Action: Take a snapshot of your RDD and dump it to a conventional data stucture that you can manipulate within your driver’s script.
* Good UseCase: Take the final result that you can actually manage on 1 computer and doing some further output on them.

- Count

* Action: Count total of row in RDD(s)

- Count by value

* Action: Count Distinct

- Take

* Action: Take top few result

- Reduce

* Action: Combine element of RDDs base on their keys

**Example:**

Lines of text from a book that we want to count how many times each word appears in RDD  Unique Key | SUM (Also can do with count by value).

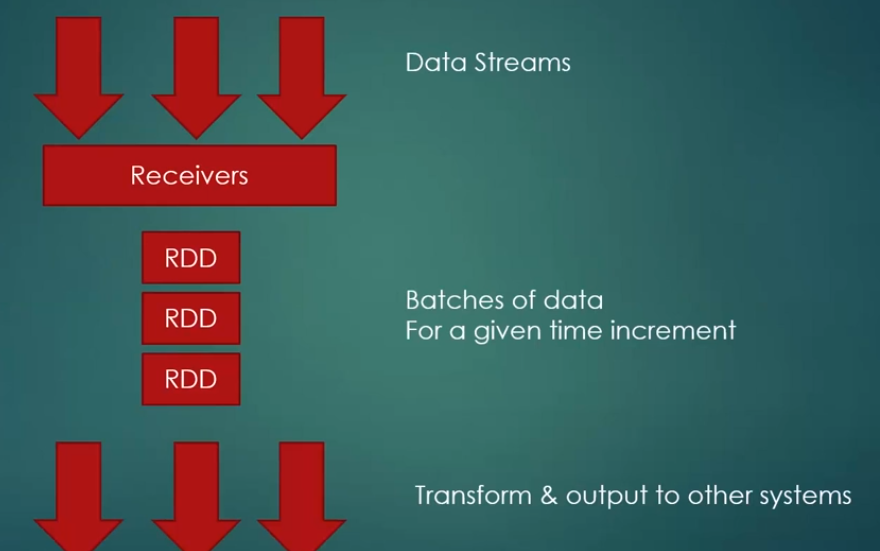
Just Like MR, we have Map in the Transforming Stage, we have the Reduce in the Action Stage.

1. **Lazy evaluation:**

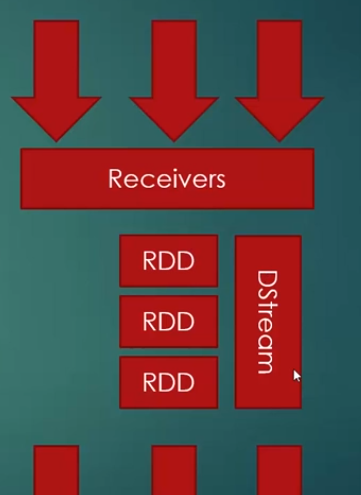
* Nothing actually happens in your driver program until an action is called.

1. Spark Streaming
   1. **Why do we use Spark Streaming:**

* Big data never stop.
* Analyze Stream of data in real time, instead of huge batch jobs daily.
* Analyzing web log to react to user behavior.
* Analyzing stream of real-time sensor data for IOT stuff.
  1. **Spark Streaming high level overview:**



* Receiver listen for that data, its break that data stream up to small RDDs, we keep transforming little RDD over and over again => external database, etc.
* Process of little RDD can happen in parallel on different worker nodes.
  1. **DStream (Discretized Stream):**
* Logical stream of information that go on and on forever is called Dstream in Spark Streaming.

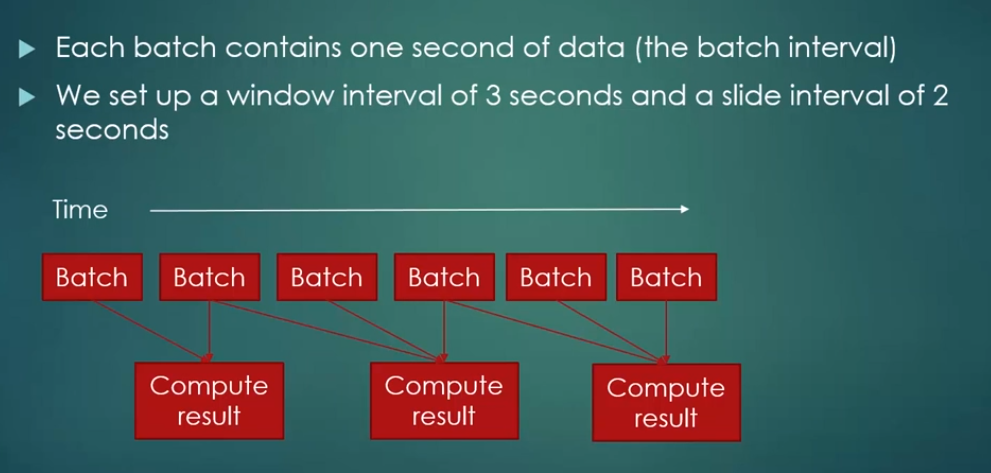


* That Stream is broken up into small RDD
* Apply transform to Dstream as a whole instead of small RDD
* Generate RDD for each time step, produce output at each time Step.
* Can be transformed and acted on in much the same way as RDD.
* You can access underlying RDD’s if you need them.
* Stateless transformation on Dstream:
  + Map
  + Flatmap
  + Filter
  + Reduce by key
  1. **Windowing:**

Allow you to compute results across a longer time period than your batch interval

* Batch interval: how often data is suck into Dstream
* Slide interval: how often a window transformation is computed
* Window interval: how far back in time the windowed transformation goes

**Example:**



* Batch interval setup:

Val SSC = new StreamingContext (“Local[\*]”, “<NAME>”, <SECOND>)

* Window interval and Slide interval setup:

Val <NAME> = <OtherValName>.reduceByKeyAndWindow(<FUNCTION>, Seconds(X), Seconds(Y))

Second(X) = Window interval

Second(Y) = Slide interval

* 1. **Spark Streaming Fault-Tolerent:**
* Incoming data is replicated to at least 2 worker nodes
* A checkpoint directory can be used to store state in case we need to restart the stream.
* What happen if receiver fail?
  + Some receivers are better than others.
  + Lost data while down
  + Receiver based on replicated, reliable data source is more resilient:
    - HDFS
    - Directly-consumed Kafka
    - Pull-based Flume
* Driver Script failure:
  + Driver script run on master node is actually a single point failure
  + Use check point directory and streaming context GetorCreate (Input is a checkpoint directory).

1. Introduction to scala
   * 1. Definition

* **Scala** is a general-purpose programming language providing support for functional programming and a strong static type system like Java.
  + 1. Learning scala through wordsheet

- Go to my github: <https://github.com/tuanjggaa/Spark-SparkStreaming-Scala>

- Download SC1-4 File and learn by yourself.

- The Syntax of Scala very similar to pascal.

4. rghunrhti

1. **How to Install Apache Spark with Scala:**

* **For Windows:**
* In stall JDK 8 (DO NOT USE 9 or higher) and keep track of where you installed the JDK.
* Download a **pre-built** version of **Apache Spark 2.3 (do not use Spark 2.4; it has a bug and only work with Linux)** from
* If necessary, download and install WinRAR so you can extract the .tgz file you downloaded. <http://www.rarlab.com/download.htm>l
* Extract the Spark archive and copy its **contents** into **C:\spark**after creating that directory. You should end up with directories like c:\spark\bin, c:\spark\conf, etc…
* Download winutils.exe from [https://sundog–s3.amazonaws.com/winutils.exe](https://sundog-spark.s3.amazonaws.com/winutils.exe) and move it into a **C:\winutils\bin** folder that you’ve created. (note, this is a 64-bit application. If you are on a 32-bit version of Windows, you’ll need to search for a 32-bit build of winutils.exe for Hadoop.)
* Open the **c:\spark\conf** folder, and make sure “File Name Extensions” is checked in the “view” tab of Windows Explorer. Rename the log4j.properties.template file to log4j.properties. Edit this file (using WordPad or something similar) and change the error level from INFO to ERROR for log4j.rootCategory
* Right-click your Windows menu, select Control Panel, System and Security, and then System. Click on “Advanced System Settings” and then the “Environment Variables” button.
* Add the following new USER variables:

1. SPARK\_HOME c:\spark
2. JAVA\_HOME (the path you installed the JDK to in step 1, for example C:\ProgramFiles\Java\jdk1.8.0\_101)
3. HADOOP\_HOME c:\winutils

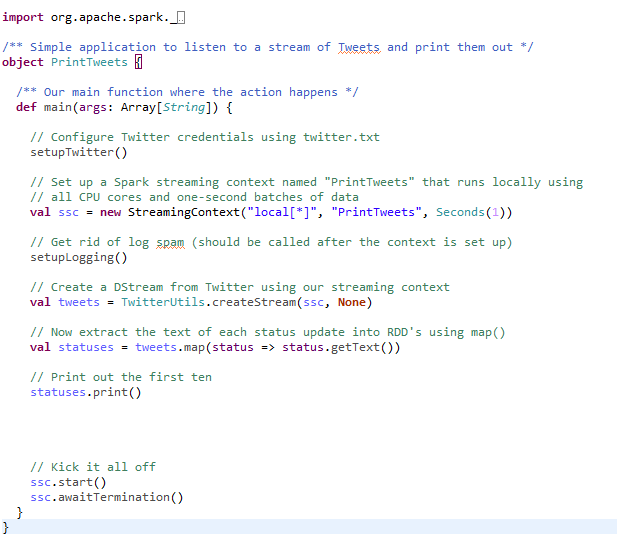
* Add the following paths to your PATH user variable:

1. %SPARK\_HOME%\bin
2. %JAVA\_HOME%\bin

* Install the latest **Scala IDE** from <http://scala-ide.org/download/sdk.html>
* Test it out:

1. Open cmd with admin mode
2. Enter **cd c:\spark**
3. Enter **dir**
4. Look for readme.md
5. Enter **Spark-shell**
6. Enter val rdd = sc.textFile(“README.md”), Enter rdd.count()
7. You Should get a count of the number of lines in that file!
8. Hit **control-D** to exit the spark shell, and close the console window
9. **Trying Some Spark Streaming Project with Scala:**

* **PrintTweets.scala**



* **Utilities.scala:**



* First thing you have to do is register for a twitter developer account
* Put the Twitter consumerKey, consumerSecret, accessToken, accessTokenSecret into a text file template like this:

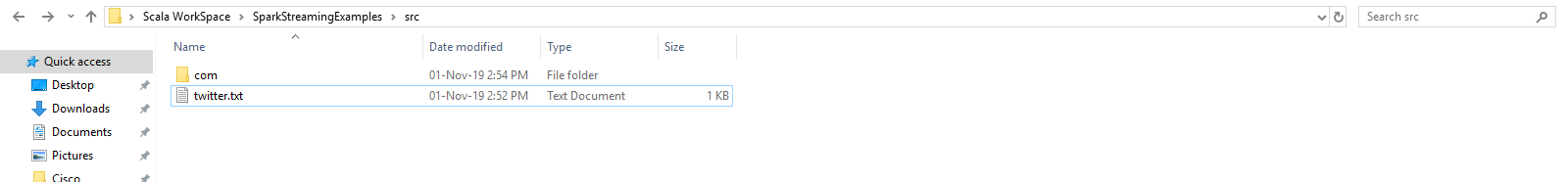
consumerKey “YOUR KEY”

consumerSecret “YOUR KEY”

accessToken “YOUR KEY”

accessTokenSecret “YOUR KEY”

* Put the Text file in your project.



* Import the **PrintTweets.scala and Ultilities.scala** into your project
* The setupLogging() and setupTwitter() function is use for Setting up your connection to twitter API and setting up the logging level to ERROR only.
* Revisiting my github and download 3 jar files. You’ll need these for access twitter API and using spark Dstream
* Move to the **PrintTweets.scala** file. You can see the Line:

val ssc = new StreamingContext("local[\*]", "PrintTweets", Seconds(1))

This means: Set up a streaming context and run this on my local computer, for every core I have in my CPU, spin up a process and run this. Allow every batch of data come in every one second.

* Next Line:

val tweets = TwitterUtils.createStream(ssc, None)

* Create a DStream from Twitter using our streaming context. This tweets will return a Json based result:

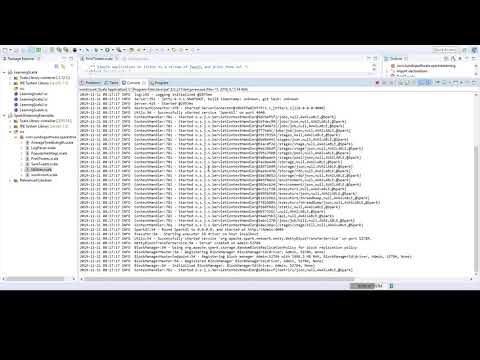
StatusJSONImpl{createdAt=Mon Nov 11 08:11:52 ICT 2019, id=1193697819510681600, text='@DangerFox Best shape',…}

* So we need to create a new Type that contains only the Text:

val statuses = tweets.map(status => status.getText())

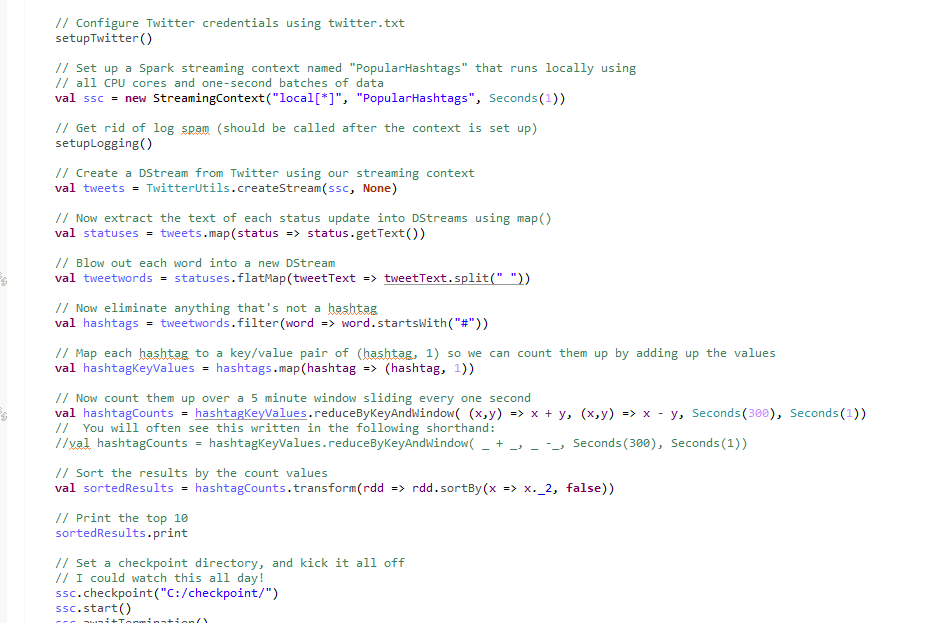
* Then we print the result.
* NOTE: You can print the result.getText() too but for the sake of simplicity let’s create a new variable that contains only Text.

**Result:**

**[](https://www.youtube.com/watch?v=EPewm4zA1ic)**

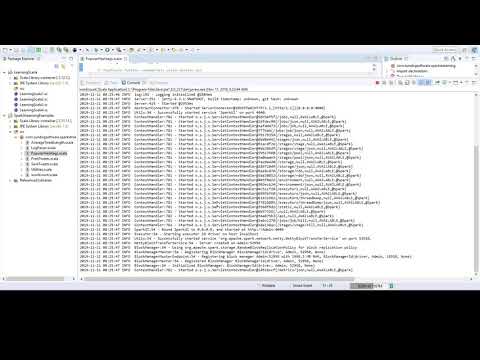
*Tracking Popular Hashtag with Spark Streaming*

* **PopularHashtags.scala:**



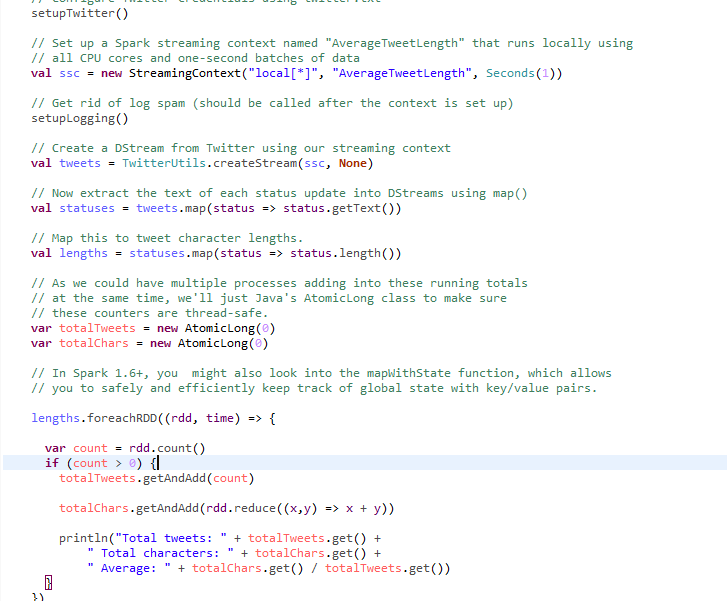
* Just like the previous program the only thing different here is we need to specify the Window interval and Slide Interval.

**Result:**

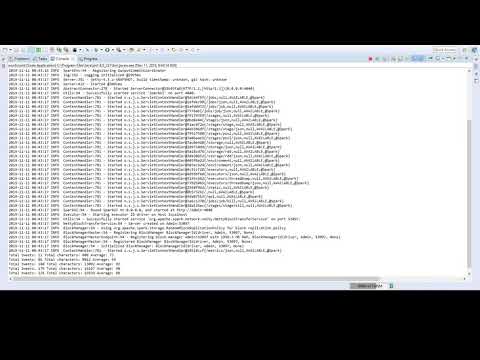
[](https://www.youtube.com/watch?v=lec6Toio9EQ)

*Tracking AVG tweets length*

* **AverageTweetLength.scala:**

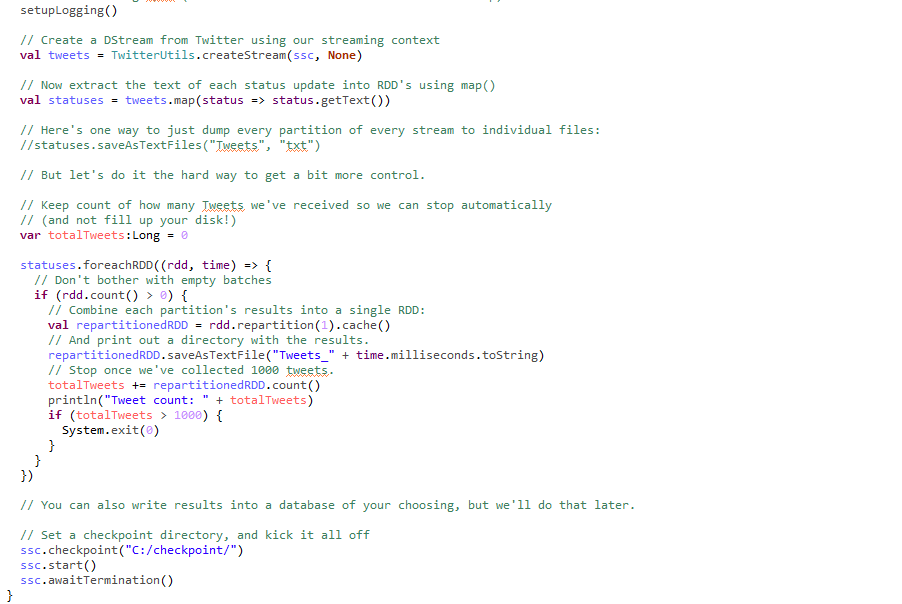


**Result**

**[](https://www.youtube.com/watch?v=N4rQAKa5Q_Y)**

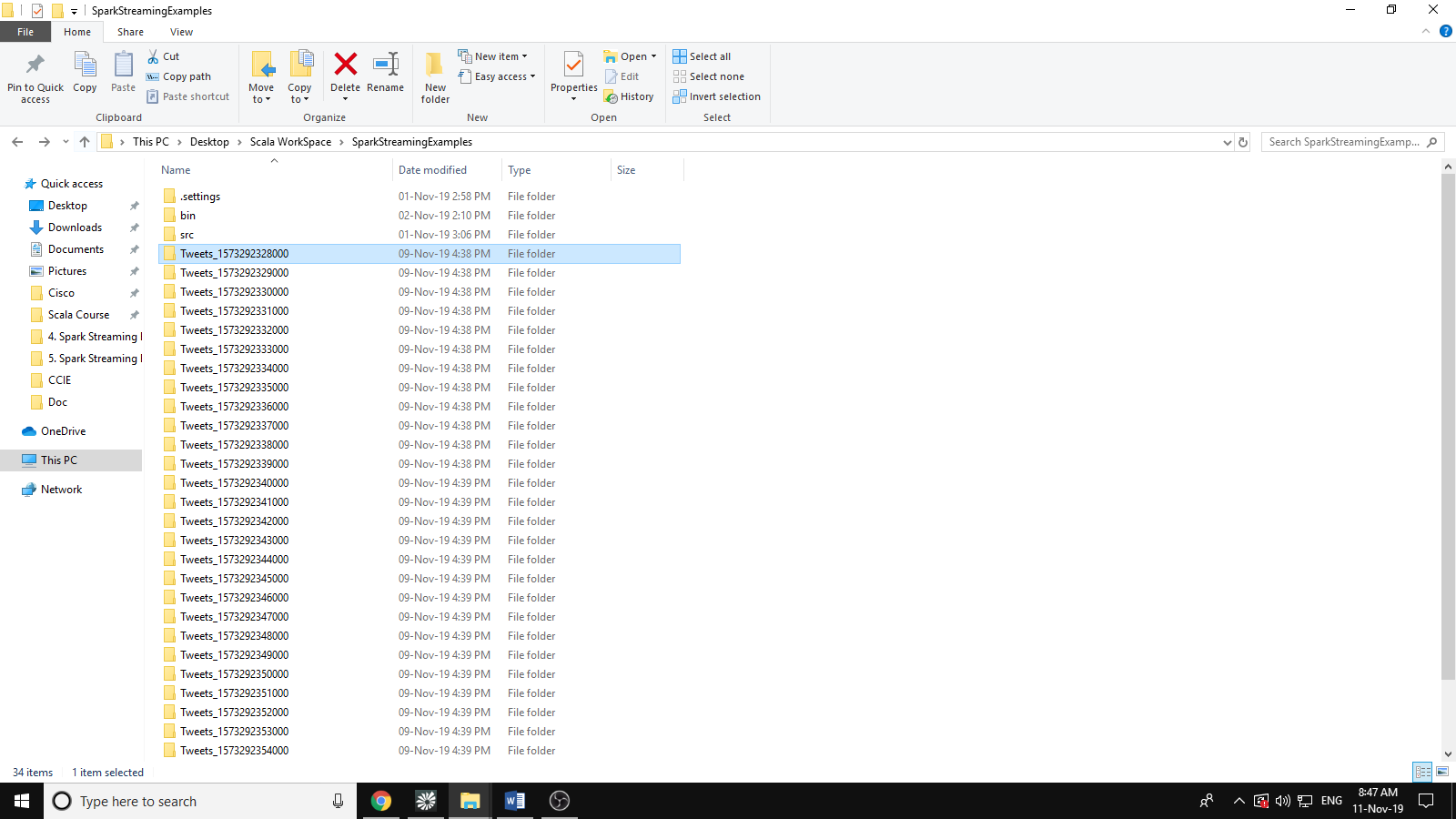
*Saving Tweets to Disk*

* **SaveTweets.scala:**

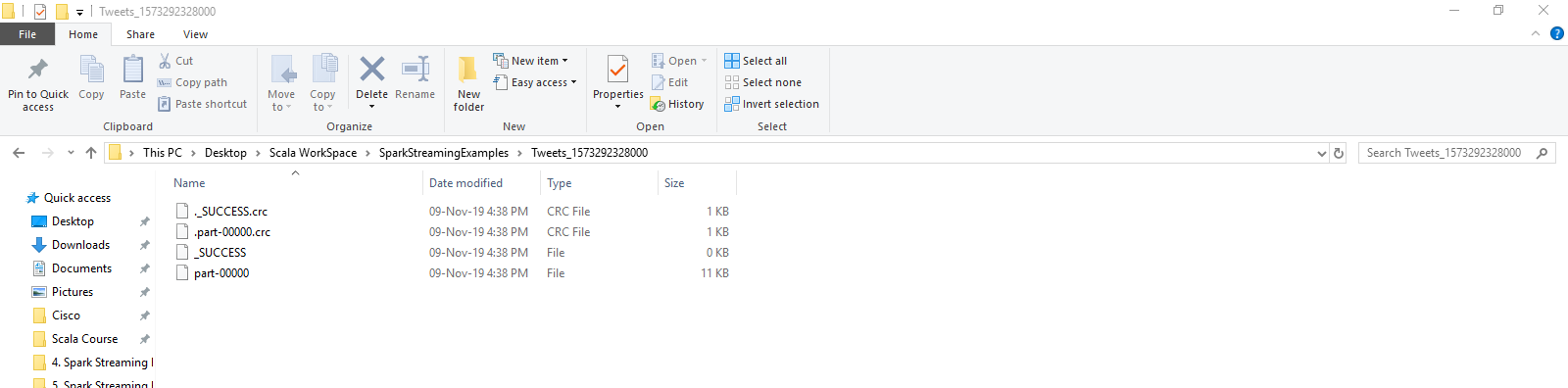


* Just like print tweet, but you need to add some code.

**Result:**



* Let check out the first batch:



* Open the part-00000 file:



1. **References:**

* Type here