**What is Spark**

A fast and general engine for large-scale data processing

Unlike MapReduce, it’s a lot easier to use spark

Write simple driver scripts to control how these big data sets are transformed and output into different systems.

Driver program: how to transform it, how to process it, how to output it. – All live within the driver program on your master node.

Driver program: create spark context – that create all the different components that you need to manipulate the data

Spark-context will communicate with your cluster manager if you area using a cluster(Spark, YARN, Mesos)

Spark maybe a next big thing that replace MapReduce in Hadoop

We can run spark on top of hadoop cluster manager(YARN)

Spark also have its own cluster manager built-in.

Cluster manager is responsible for controlling the processing of your data across entire cluster

Many Node that are running their own executors and there could potentially be multiple executors in 1 machine => Horizontal scalable – Add more and more server and process more and more data as a result, never have to bound by the memory of a single machine.

It’s Fast, 100x faster than MR in memory and 10x time on disk.

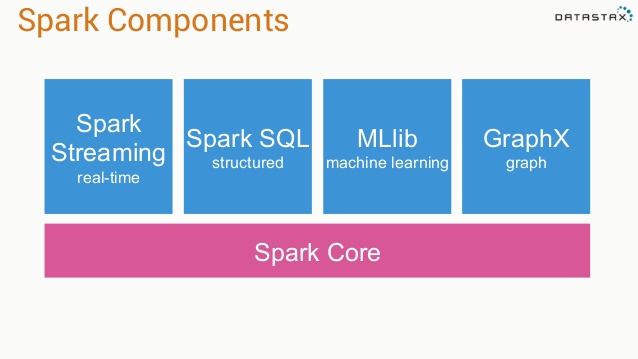
USE DAG ENGINE( directed acyclic graph ) to optimizes workflows – Nothing Actually happen until you tell spark to go and do it. => spark go off and said what’s the optimal way to approach this problem as a whole.

Many Company use it. => It’s hot

Code in JAVA, SCALA, PYTHON

Spark is built around RDD

Components of Spark



Every thing is built on top of spark core which contains all the guts of actually creating, transforming performing action on RDD.

SPARK Streaming – let 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

**RDD** – 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 datastorage.

Dataset – It’s a dataset

**Spark Context**

Created by your driver program.

Responsible for making RDD resilient and distributed

Whenever we start with any spark program, we’ve got to create a spark context first.

Spark context object is what create our rdd objects. It’s like the mastermind behind the whole script

In SparkStreaming, you create a streaming context – Just a special spark context that can deal with real time data

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.

We can also create RDD from data that’s in memory

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

Also create RDD from textfile

Sc.textFile(“PATH”)

We can also create from Distributed file System – real world

Can Also create from:

JDBC

Cassandra

Hbase

…

Json

…

**Transforming RDD**

Once we have RDD that represent some big dataset, what do we do with it.

* We Transform it or we do some action with it.
* 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)

**RDD Actions**

* Collect (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 (rows)
* Count by value (Count Distinct)
* Take (take top few result)
* Reduce (combine element of RDD base on their keys)

Ex: Lines of text from a book that we want to count how many times each word appear in RDD => Unique Key | SUM (Also can do with county by value)

* …
* Just Like MR, we have Map in the Transforming Stage, We have the Reduce in the Action Stage

**Lazy evaluation**

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

**Why Spark Streaming**

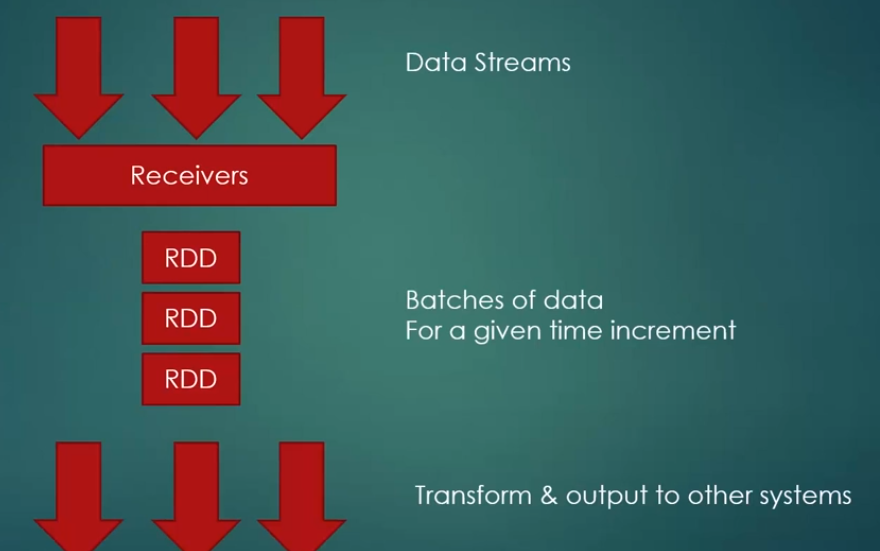
Big data never stop

Analyze dt stream 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

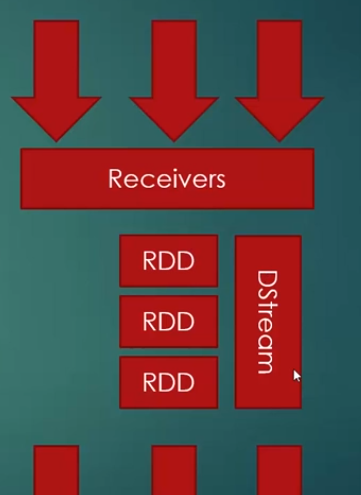
**Spark Streaming High level**



Receiver listen for that data, its break that data stream up to small rdds, we keep transforming little rdd over and over again => external db,…

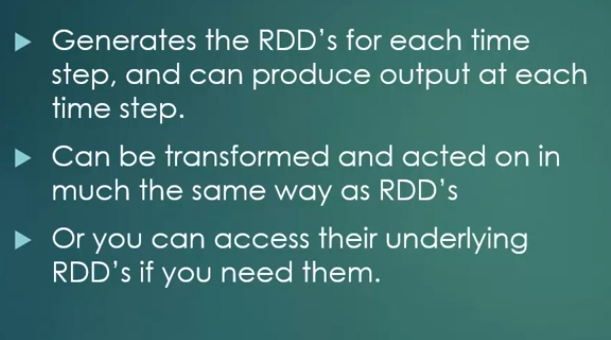
* Process of little rdd can happen in parallel on different worker nodes.

**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



**Stateless transformation on Dstream**

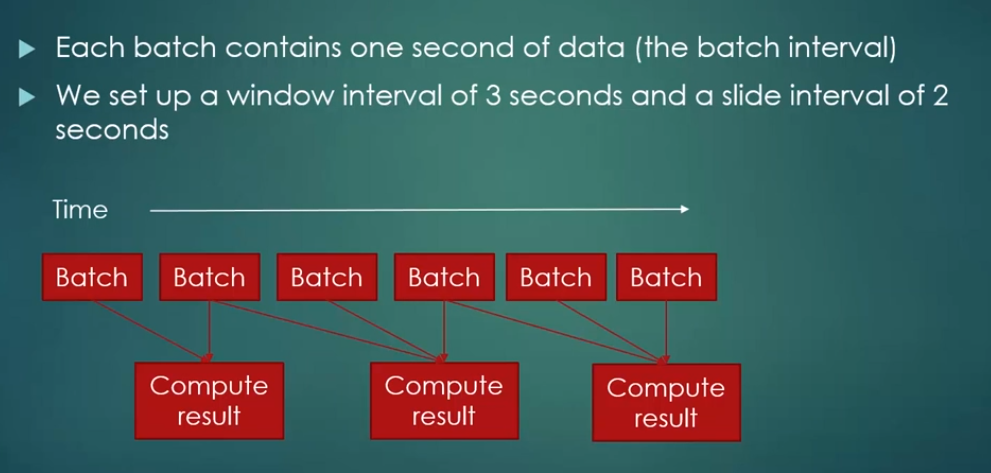
* Map
* Flatmap
* Filter
* Reduce by key

**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

Ex:



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

**Spark Streaming Fault-Tolerent**

Incoming data is replicated to at least 2 worker nodes

A checkpoint directory can be use 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)

**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 works with Linux)**from <https://spark.apache.org/downloads.html>
* If necessary, download and install WinRAR so you can extract the .tgz file you downloaded. <http://www.rarlab.com/download.htm>
* 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 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:

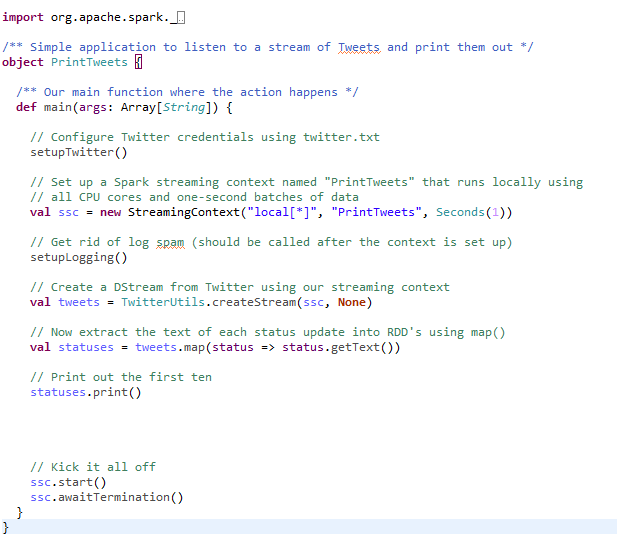
1. Open cmd with admin mode
2. Enter **cd c:\spark**
3. Enter **dir**
4. Look for readme.md
5. Enter **Spark-shell**
6. Enter val rdd = sc.textFile(“README.md”), Enter rdd.count()
7. should get a count of the number of lines in that file!
8. Hit **control-D** to exit the spark shell, and close the console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

**Learning Scala through sc worksheet(.sc)**

* Go to my github: <https://github.com/tuanjggaa/Spark-SparkStreaming-Scala>.
* Download SC1-4 File and learn by your self.
* The Syntax of Scala very similar to pascal, or I think so.

**Trying our First Scala Project: Print Tweets**

**PrintTweets.scala**



**Utilities.scala**



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

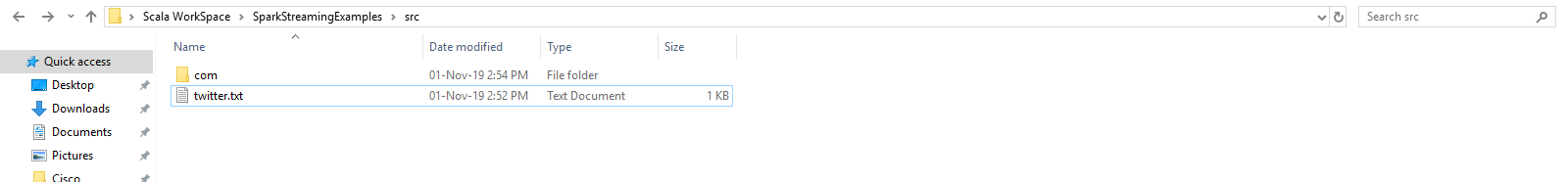
consumerKey “YOUR KEY”

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 file. 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:

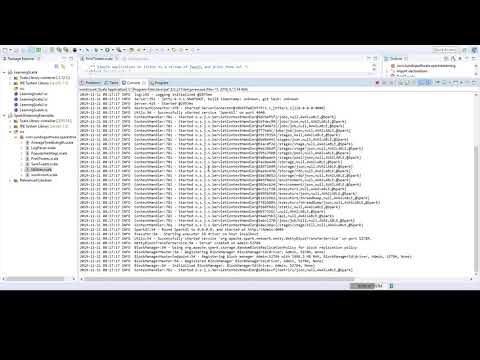
StatusJSONImpl{createdAt=Mon Nov 11 08:11:52 ICT 2019, id=1193697819510681600, text='@DangerFox Best shape',…}

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

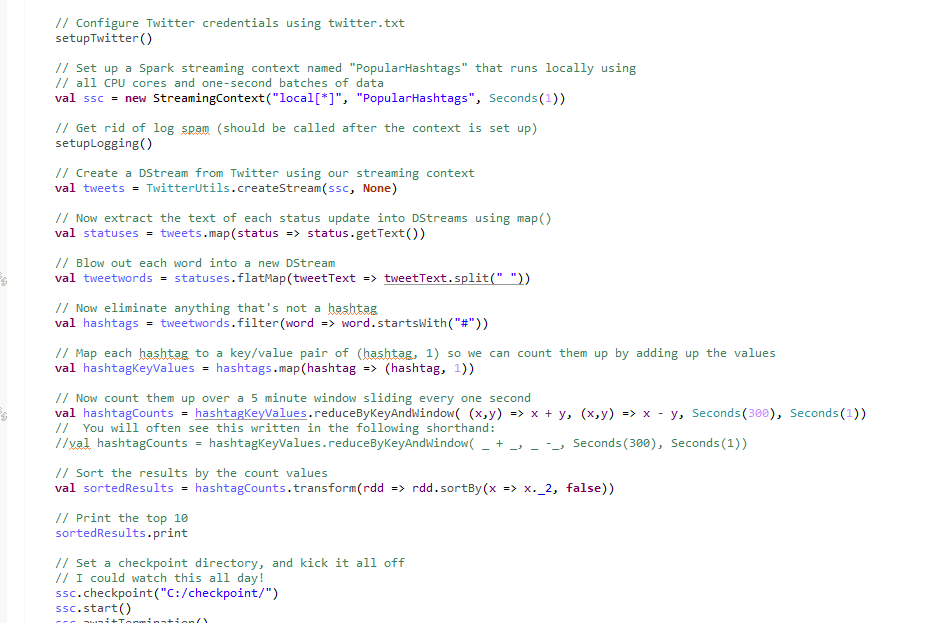
val statuses = tweets.map(status => status.getText())

* 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:**

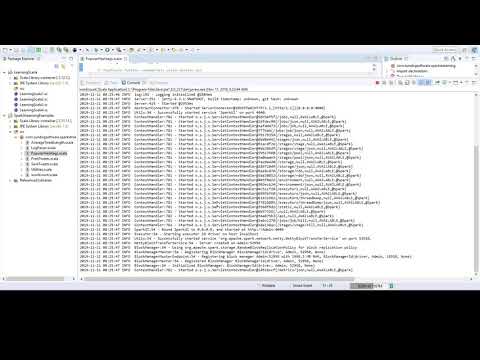
**[](https://www.youtube.com/watch?v=EPewm4zA1ic)**

**Tracking Popular Hashtag with Spark Streaming**

**PopularHashtags.scala**

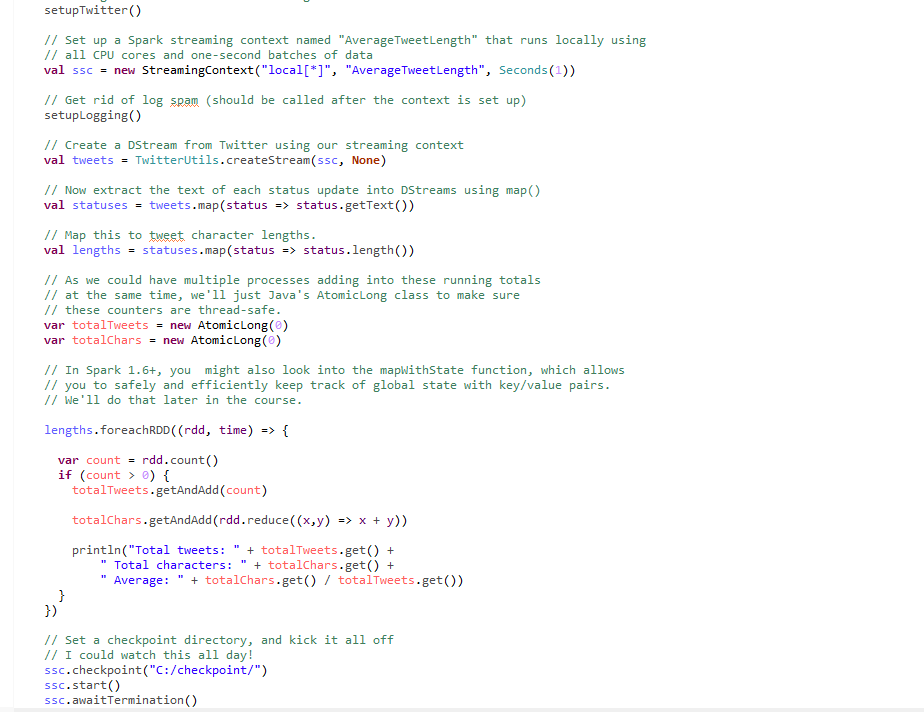
Just like the previous program the only thing different here is we need to specify the Window interval and Slide Interval.

**Result:**

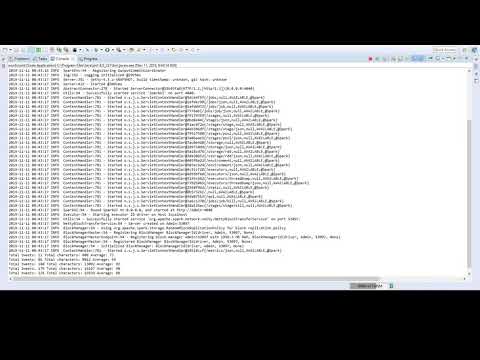
[](https://www.youtube.com/watch?v=lec6Toio9EQ)

**Tracking AVG tweets length**

**AverageTweetLength.scala**

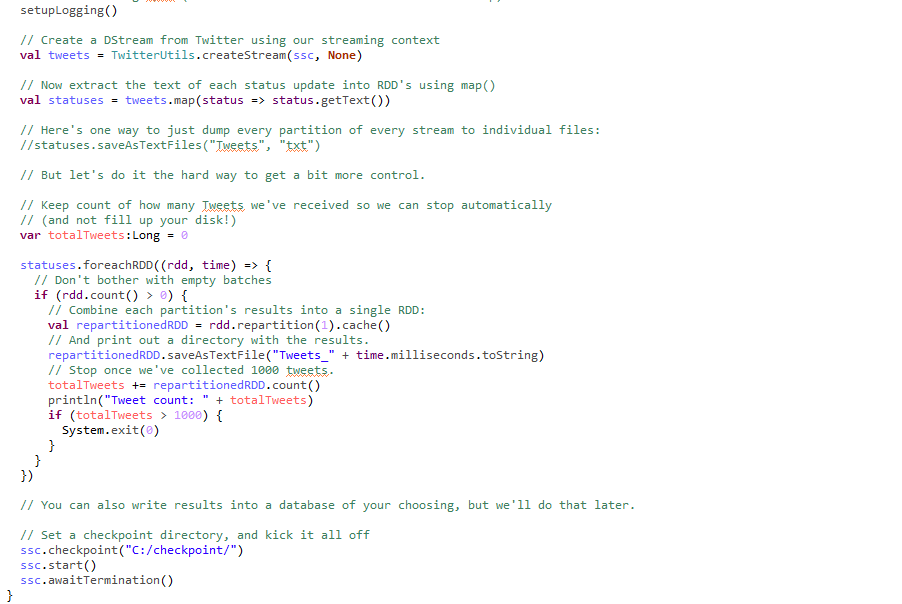


**Result**

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

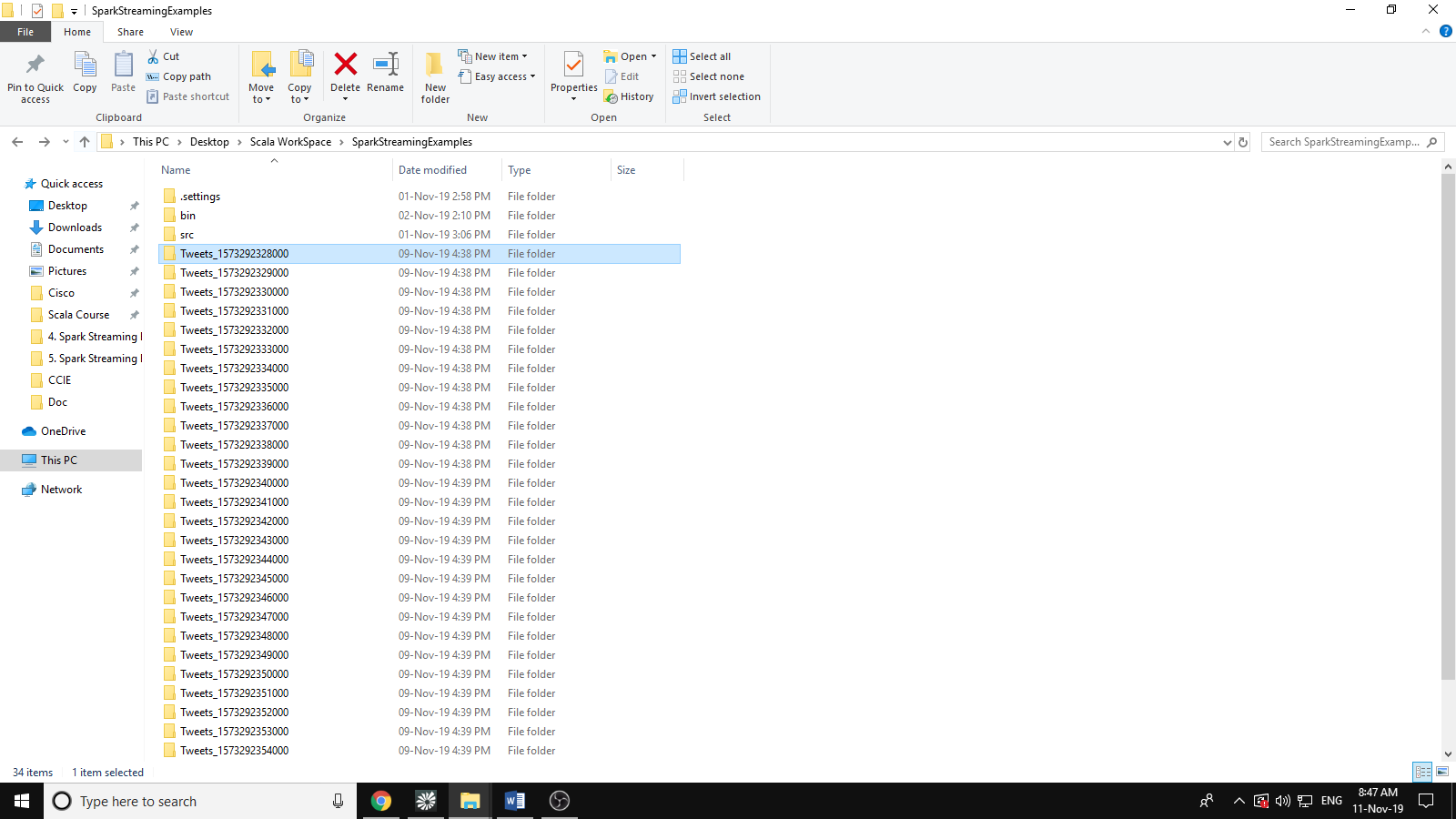
**Saving Tweets to Disk**

**SaveTweets.scala**

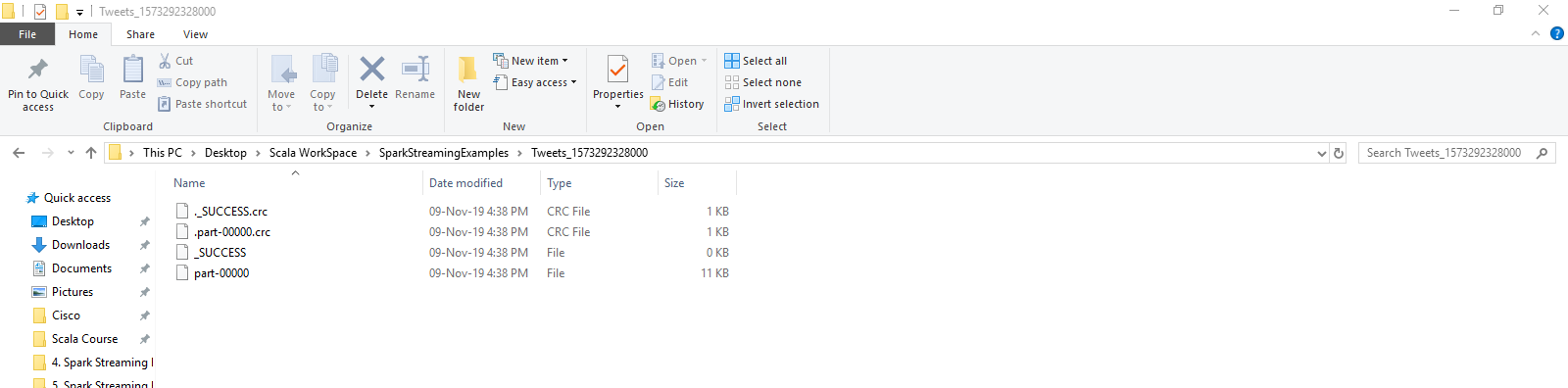


Just like print tweet, but you need to add some code:

**Result:**



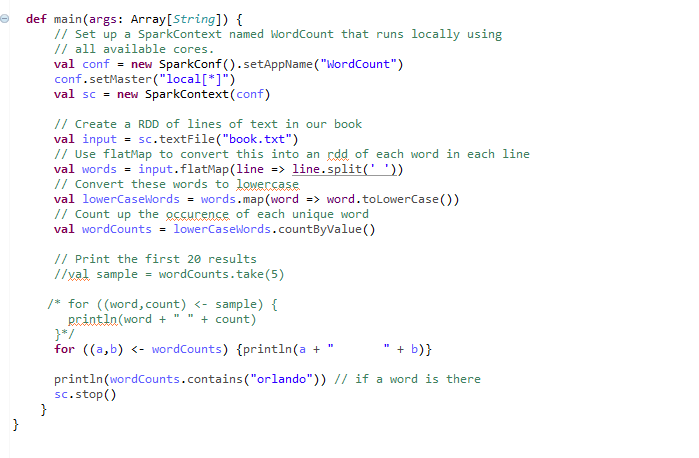
* Let check out the first batch



* Open the part-00000 file:



**WordsCount with files:**



**Result:**

