## [EXPEDIA GROUP TECHNOLOGY — DATA](https://medium.com/expedia-group-tech/data/home)

# **Apache Spark Structured Streaming — Input Sources (2 of 6)**

## Getting into Spark Streaming with Rate, Socket, File, and Kafka Input Sources

[IMG_256](https://medium.com/@bhadani.neeraj.08?source=post_page-----6a72f798838c--------------------------------)

[IMG_257](https://medium.com/expedia-group-tech?source=post_page-----6a72f798838c--------------------------------)

[Neeraj Bhadani](https://medium.com/@bhadani.neeraj.08?source=post_page-----6a72f798838c--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https://medium.com/_/subscribe/user/f9ed76fbd13a&operation=register&redirect=https://medium.com/expedia-group-tech/apache-spark-structured-streaming-input-sources-2-of-6-6a72f798838c&user=Neeraj+Bhadani&userId=f9ed76fbd13a&source=post_page-f9ed76fbd13a----6a72f798838c---------------------post_header-----------)

Published in

[Expedia Group Technology](https://medium.com/expedia-group-tech?source=post_page-----6a72f798838c--------------------------------)

·

8 min read

·

Feb 4, 2021

104



Photo by Viktor Talashuk on Unsplash

In [part 1](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-first-streaming-example-1-of-6-e8f3219748ef), we discussed an overview of Apache Spark™️ Structured Streaming with a very basic example using the rate input source. In this blog, we discuss further ****input sources**** in detail using examples.

You may also be interested in some of my earlier posts on Apache Spark.

* [Apache Spark Structured Streaming — First Streaming Example](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-first-streaming-example-1-of-6-e8f3219748ef) (Part 1 of this series)
* [Deep Dive into Apache Spark DateTime Functions](https://medium.com/expedia-group-tech/deep-dive-into-apache-spark-datetime-functions-b66de737950a)
* [Working with JSON in Apache Spark](https://medium.com/expedia-group-tech/working-with-json-in-apache-spark-1ecf553c2a8c)
* [Deep Dive into Apache Spark Window Functions](https://medium.com/expedia-group-tech/deep-dive-into-apache-spark-window-functions-7b4e39ad3c86)
* [Deep Dive into Apache Spark Array Functions](https://medium.com/expedia-group-tech/deep-dive-into-apache-spark-array-functions-720b8fbfa729)
* [Start Your Journey with Apache Spark](https://medium.com/expedia-group-tech/start-your-journey-with-apache-spark-part-1-3575b20ee088)

# **Input Sources**

Spark Streaming ingests data from different types of input sources for processing in real-time.

* ****Rate**** (for Testing): It will automatically generate data including 2 columns timestamp and value . This is generally used for testing purposes. We demonstrated this in [part 1 of this series](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-first-streaming-example-1-of-6-e8f3219748ef).
* ****Socket**** (for Testing): This data source will listen to the specified socket and ingest any data into Spark Streaming. It is also used only for testing purposes.
* ****File****: This will listen to a particular directory as streaming data. It supports file formats like CSV, JSON, ORC, and Parquet. You can find the latest supported file format list [here](https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html" \l "input-sources" \t "https://medium.com/expedia-group-tech/_blank).
* ****Kafka****: This will read data from Apache Kafka® and is compatible with Kafka broker versions 0.10.0 or higher

It’s time to get our hands dirty. Let's ingest data from each of the input sources.

IMG_259

# **Input Sources — Socket**

## **Setup**

Import libraries and create Spark Session

## **Create Streaming DataFrame**

Create Streaming DataFrame using socket source. Also, check if DataFrame isStreaming.

We use the socket format to read data from a socket (127.0.0.1:9999). You can use any arbitrary permitted port to listen.

Output:

Streaming DataFrame : true

## **Transformation: Word Count**

Here we count words from a stream of data coming from this socket. Also, we check the schema of our streaming DataFrame.

Output:

Schema of DataFame wordCount.  
root  
 |-- words: string (nullable = true)  
 |-- count: long (nullable = false)

## **Output to Console**

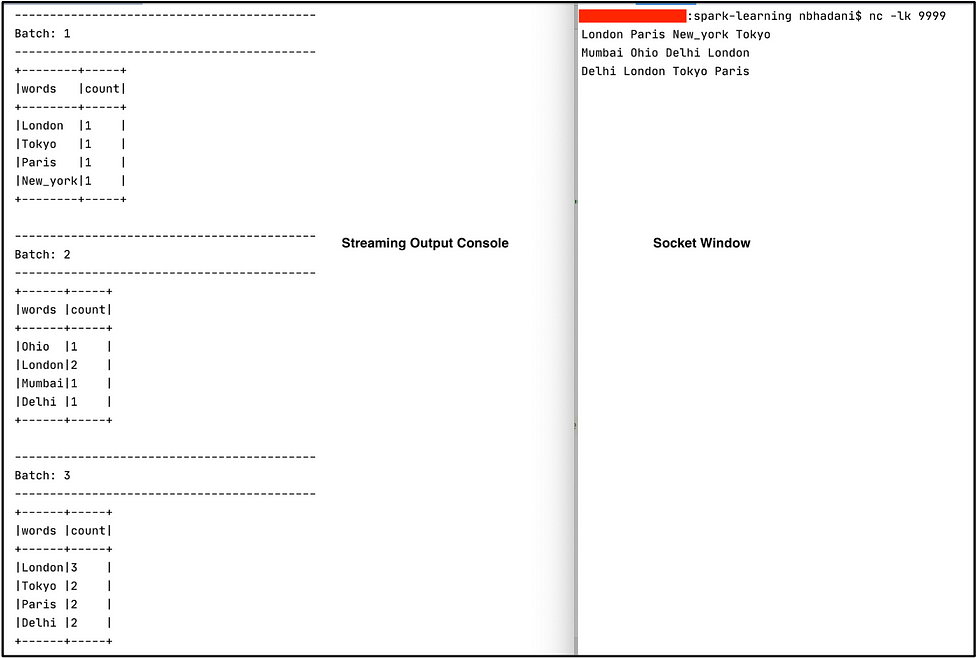
Print the contents of streaming DataFrame wordCount on console.

Open the port 9999 on localhost(127.0.0.1) and send some data to count. We use the [netcat](http://netcat.sourceforge.net/" \t "https://medium.com/expedia-group-tech/_blank) utility to open the port. Open a terminal and run the command below.

nc -lk 9999

Start the streaming application and send data to the port.

*Note: Make sure you have the socket open before you start the streaming application.*



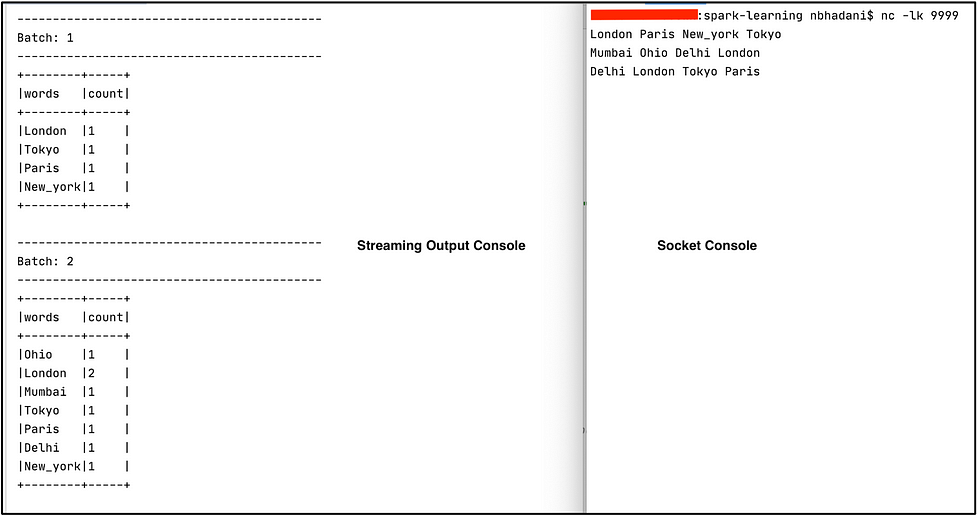
On the right window we send data to our socket and on the left our Spark Streaming application prints to the console. We get a word count on the output console of the data sent to the socket. Since we are running our Spark Stream application in update output mode, we see that only records that are updated in a particular batch are output to the console. From the first line, London Paris New\_york Tokyo, we have the output in Batch:1. From the second line, Mumbai Ohio Delhi London, we have the output in Batch:2 (Note that we sent the second line only once the first line was fully processed). Since Mumbai Ohio Delhi are new records and our streaming application has seen London for a second time, we have got the count as 2 for London and 1 for rest of the words in second line Mumbai Ohio Delhi . Also, the words from the first line which don’t appear in second line are not printed on output of Batch:2. Again this is because we are working in update output mode which only prints updated records.

Try outcomplete output mode as well and see how it works.

*Note: kill the process running on port 9999 to start again for complete mode. Or use a different port.*

// Find the process id using below command on terminal.  
ps | grep 9999// kill process using below command.  
kill -9 <p\_id>

Again, open the port using the nccommand and start the streaming application in complete mode.



On the right window, we send data to our socket and the left our Spark Streaming application prints. Batch:1 and Batch:2represent the output for line-1 and line-2 respectively. In complete mode, for Batch:2 we get all the records our streaming application has seen so far. For e.g. Tokyo is only present in line-1 but not in line-2. However, we get Tokyo in Batch:2 as well because complete mode writes all the records it has processed so far.

*Note: Output for line-3 is trimmed out of the screenshot.*

For easy reference, you can find the complete code on ***[GitHub](https://github.com/NeerajBhadani/spark-streaming/blob/master/src/main/scala/streamSocketSource.scala" \t "https://medium.com/expedia-group-tech/_blank)***.

# **Input Sources — File**

With file input source, our application will wait for available data in the specified directory. We will use some of the stock data available [here](https://github.com/szrlee/Stock-Time-Series-Analysis/tree/master/data" \t "https://medium.com/expedia-group-tech/_blank). For example, Apple stock data present in this file: [AAPL\_2006–01–01\_to\_2018–01–01.csv](https://github.com/szrlee/Stock-Time-Series-Analysis/blob/master/data/AAPL_2006-01-01_to_2018-01-01.csv" \t "https://medium.com/expedia-group-tech/_blank). We will take the data for a few years like 2015, 2016, and 2017 and manually save it to a different file like AAPL\_2015.csv, AAPL\_2016.csvand AAPL\_2017.csv respectively. Similarly, we will create the sample data for Google, Amazon, and Microsoft as well. We will keep all the CSV files locally under data/stocks folder. Also, create another folder data/stream which we will use to simulate the streaming data.

## **Setup**

## **Schema**

Our data contains the fields Date,Open,High,Low,Close,Adj Close,Volume and we will extract Name from the filename using a custom function.

Here we define the schema and write a custom function to extract the stock ticker symbol.

## **Create Streaming DataFrame**

We create a streaming DataFrame to read csv data from a specified directory data/stream and apply the above schema . We also specify the param maxFilesPerTrigger = 2 , which means our application will process a maximum of 2 CSV files in each batch. At the end, we create another column called Name using the function getFileName. That column contains stock ticker symbols like GOOGL, AMZN, AAPL etc.

## **Transformation**

Perform basic aggregation on our streaming DataFrame.

We group the data based on stock Name, Year and find the maximum value of the HIGH column.

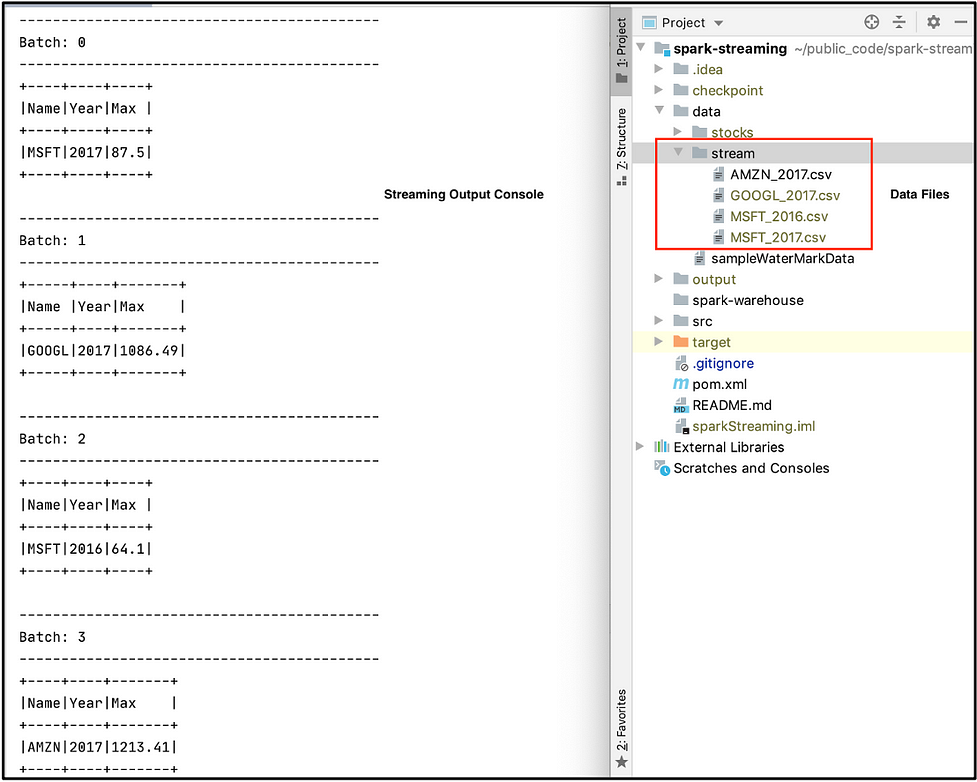
We can also perform the above transformation using a SQL query. In this code sample, we register the streaming DataFrame as a temporary view and execute a SQL query on it.

## **Output to Console**

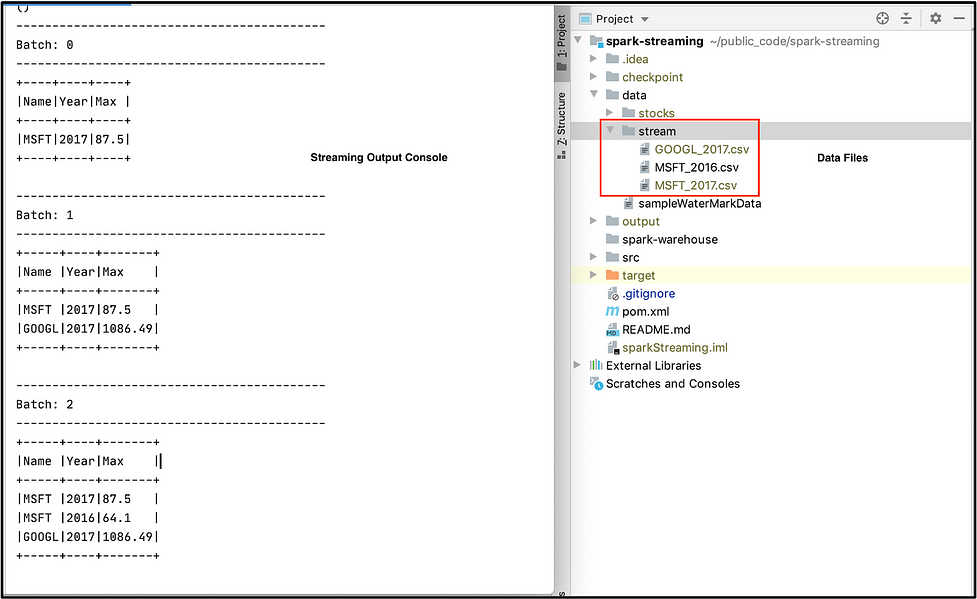
Print the contents of streaming DatFrame to console using update mode.

Let’s start our streaming application now, it waits for data in the data/stream folder. Copy files in the sequence below from data/stocks to data/stream to simulate streaming.

1. MSFT\_2017.csv
2. GOOGL\_2017.csv
3. MSFT\_2016.csv
4. AMZN\_2017.csv



First we moved the file MSFT\_2017.csv and got a max HIGH stock value in Batch: 0 for Microsoft 2017. Second, we moved GOOGL\_2017.csv and got the output for Google 2017 in Batch: 1. Third, we moved MSFT\_2016 and saw the output for Microsoft 2016 in Batch:2. Since we usedupdate mode, only updated records are output to console. Feel free to try out complete mode at your end. Please find below the output of complete mode as well.



For easy reference, you can find the complete code on ***[GitHub](https://github.com/NeerajBhadani/spark-streaming/blob/master/src/main/scala/streamFileSource.scala" \t "https://medium.com/expedia-group-tech/_blank)***.

# **Input Sources — Kafka**

We will read data from Kafka and display it on the console. In order to read data from Kafka, first we need to set up Kafka and publish messages to a Kafka topic which we will then read into Spark Streaming.

## **Kafka Setup**

You can follow below steps to setup Kafka.

* Install Kafka: We can refer to [this](https://docs.confluent.io/platform/current/quickstart/ce-quickstart.html" \t "https://medium.com/expedia-group-tech/_blank) article to install Kafka.
* Start Confluent Service

confluent local services start

* Create a Kafka Topic

kafka-topics --create \  
--zookeeper localhost:2181 \  
 --replication-factor 1 \  
 --partitions 1 \  
 --topic test

* List Kafka Topic

List Kafka Topic

* Publish data to a Kafka topic, so that we can read it back into Spark Streaming

kafka-console-producer --broker-list localhost:9092 --topic test

* Consume data from Kafka topic to verify the published data.

kafka-console-consumer --bootstrap-server localhost:9092 --topic test

## **Spark Streaming Setup**

Import required libraries and create a Spark session.

## **Create Streaming DataFrame**

Read data from Kafka into Spark Streaming.

We use the kafka format to read data from Kafka. We specify the details of our locally installed Kafka bootstrap server and subscribe to Kafka topic test created above. At last, we select only data ingested into Kafka topic which is present in the value column.

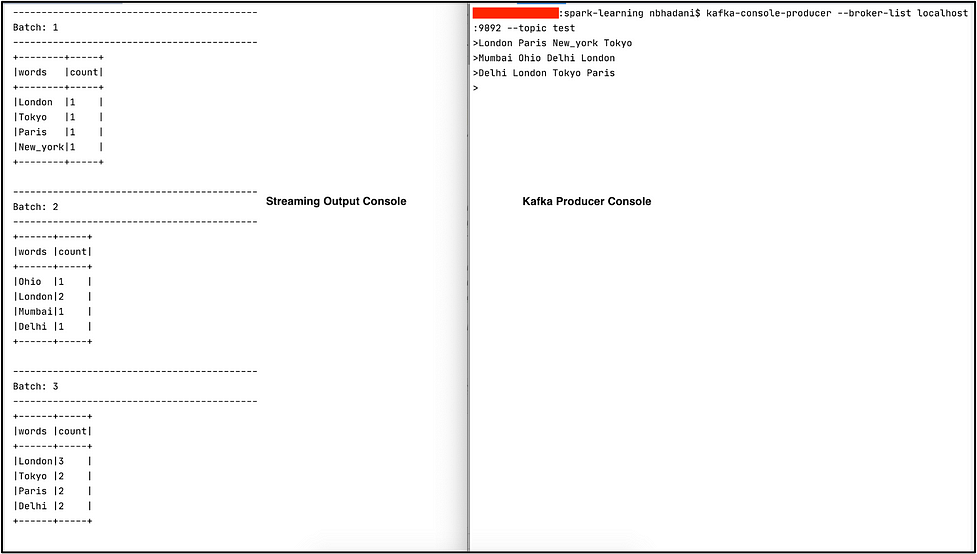
## **Transformation**

Perform simple word count similar to socket source example.

## **Output to Console**

Print the contents of streaming DataFrame to console.

Output looks very similar to our socket source example but this time our streaming application reads data from Kafka.



For easy reference, you can find the complete code on ***[GitHub](https://github.com/NeerajBhadani/spark-streaming/blob/master/src/main/scala/streamKafkaSource.scala" \t "https://medium.com/expedia-group-tech/_blank)***.

The next post in this series covers [output sinks](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-output-sinks-3-of-6-ed3247545fbc) of different types.

Here are other blogs on Apache Spark Structured Streaming series.

* [Apache Spark Structured Streaming — First Streaming Example](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-first-streaming-example-1-of-6-e8f3219748ef)
* [Apache Spark Structured Streaming — Output Sinks](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-output-sinks-3-of-6-ed3247545fbc)
* [Apache Spark Structured Streaming — Checkpoints and Triggers](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-checkpoints-and-triggers-4-of-6-b6f15d5cfd8d)
* [Apache Spark Structured Streaming — Operations](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-operations-5-of-6-40d907866fa7)
* [Apache Spark Structured Streaming — Watermarking](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-watermarking-6-of-6-1187542d779f)

IMG_265

I hope you enjoyed learning about the different input sources which can ingest data into Spark Streaming!

# **References**

* [https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#input-sources](https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html" \l "input-sources" \t "https://medium.com/expedia-group-tech/_blank)