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

# **Apache Spark Structured Streaming — Output Sinks (3 of 6)**

## Straight Outta Spark Streaming

[](https://medium.com/@bhadani.neeraj.08?source=post_page-----ed3247545fbc--------------------------------)

[IMG_257](https://medium.com/expedia-group-tech?source=post_page-----ed3247545fbc--------------------------------)

[Neeraj Bhadani](https://medium.com/@bhadani.neeraj.08?source=post_page-----ed3247545fbc--------------------------------)

·

[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-output-sinks-3-of-6-ed3247545fbc&user=Neeraj+Bhadani&userId=f9ed76fbd13a&source=post_page-f9ed76fbd13a----ed3247545fbc---------------------post_header-----------)

Published in

[Expedia Group Technology](https://medium.com/expedia-group-tech?source=post_page-----ed3247545fbc--------------------------------)

·

7 min read

·

Feb 18, 2021

91

1



Photo by Matthew Feeney on Unsplash

In previous posts of this series, we discussed an [overview](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-first-streaming-example-1-of-6-e8f3219748ef) as well as [input sources](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-input-sources-2-of-6-6a72f798838c) for Apache Spark™️ Streaming. This time, we discuss the available range of ****output sinks****.

You may also be interested in my other 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)
* [Apache Spark Structured Streaming — Input Sources](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-input-sources-2-of-6-6a72f798838c)
* [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)

IMG_259

# **Output Sinks**

In Spark Streaming, output sinks store results into external storage.

* ****Console sink****: Displays the content of the DataFrame to console. In this series, we have only used console sink, refer to [previous](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-first-streaming-example-1-of-6-e8f3219748ef) [posts](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-input-sources-2-of-6-6a72f798838c) for details.
* ****File sink****: Stores the contents of a DataFrame in a file within a directory. Supported file formats are csv, json, orc, and parquet.
* ****Kafka sink****: Publishes data to a Kafka® topic.
* ****Foreach sink****: Applies to each row of a DataFrame and can be used when writing custom logic to store data.
* ****ForeachBatch sink****: Applies to each micro-batch of a DataFrame and also can be used when writing custom logic to store data.

It’s time to get our hands dirty; let’s discuss each output sink in detail.

## **Setup**

We use the same csv data files as in the input sources post of this series to create a streaming DataFrame from a file source. Please refer to [this](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-input-sources-2-of-6-6a72f798838c) blog for more details on the code below.

# **Output sink — Console**

We covered the console output sink multiple times in [a previous post](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-input-sources-2-of-6-6a72f798838c).

# **Output sink — File**

The file sink stores the contents of a streaming DataFrame to a specified directory and format. We use initDf (created above) and apply a simple transformation before storing it to the file system.

## **Transformation**

Perform a very simple transformation by selecting a few columns from initDF

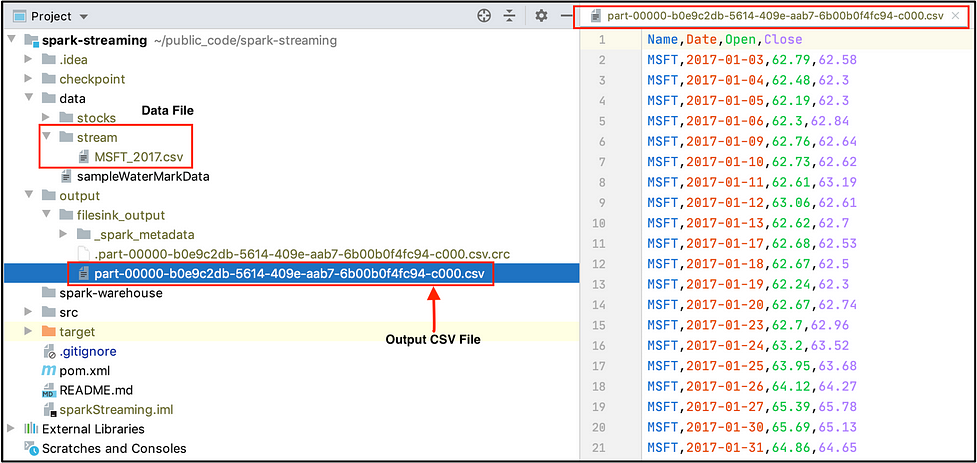
val resultDf = initDF.select("Name", "Date", "Open", "Close")

## **Output to file sink**

*Note: File sink only supports append output mode.*

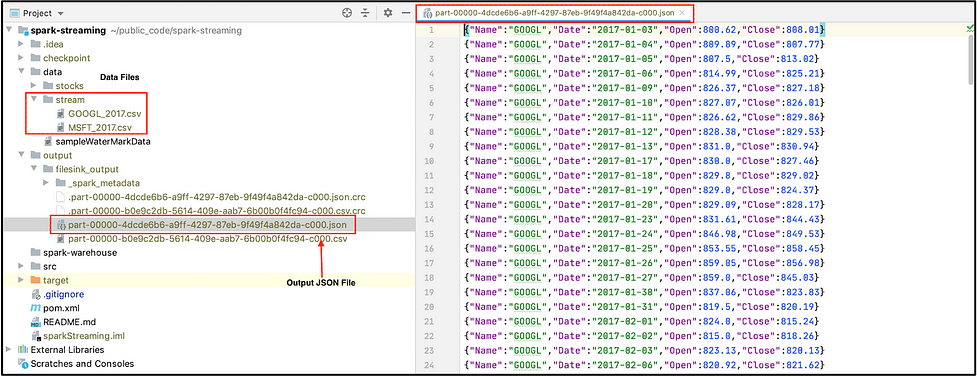
Store the content of DataFrame resultDf using file sink.

We store the results into csv files. We also use checkpoint, which we discuss in a separate blog post. Execute the code above and check the output folder output/filesink\_output .



The output of our resultDf DataFrame is stored in directory output/filesink\_output . Check out the selected columns (Name, Date, Open, Close) in the csv on the right side of the snapshot.

Now change the file format to json and review the output. The content of our resultDf DataFrame is stored in a json file.



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

# **Output sink — Kafka**

With the kafka sink, we publish the content of our streaming DataFrame to a Kafka topic using the initDf DataFrame defined in the Setup section above. You can refer to [this](https://github.com/NeerajBhadani/spark-streaming/blob/master/src/main/scala/setupKafka" \t "https://medium.com/expedia-group-tech/_blank) to install Kafka and create a Kafka topic.

## **Transformation**

Kafka expects data in a column name value, so let's concatenate all the column values and store them in one value column.

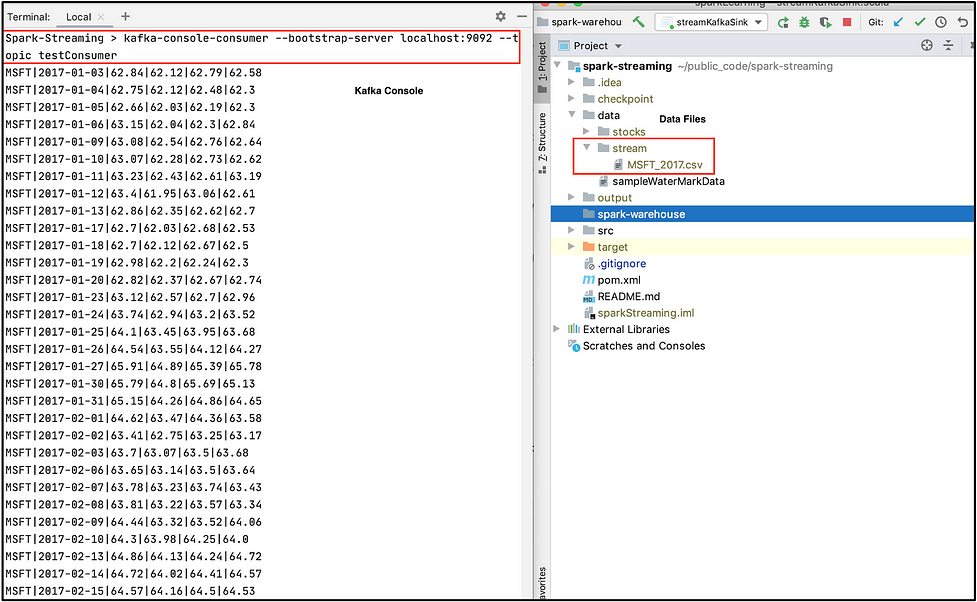
## **Output to Kafka sink**

We pass the value "kafka” to format() and specify other necessary Kafka information like "kafka.bootstrap.servers” and the "topic” to which we publish.

## **Execute our streaming application**

We use the command below to consume the data which our streaming application published to Kafka.

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



In the screenshot above, as soon as we drop a file into data/stream (a folder in the right window), we see our transformed output on Kafka’s console on the left window (which is pipe | separated). For easy reference, you can find the complete code on ***[GitHub](https://github.com/NeerajBhadani/spark-streaming/blob/master/src/main/scala/streamKafkaSink.scala" \t "https://medium.com/expedia-group-tech/_blank)***.

# **Output sink — foreachBatch**

So far we discussed sinks where the output system was already defined like file, kafka, or console. What if we would like to store data in any arbitrary storage like a NoSQL DB (for example MongoDB) or a Relational DB (like MySQL). By using foreach and foreachBatch we can write custom logic to store data. foreach performs custom write logic on each row and foreachBatch performs custom write logic on each micro-batch.

Using foreachBatch, we write each micro-batch to storage defined in our custom logic. In this case, we store the output of our streaming application to MySQL DB.

## **Transformation**

Select a few columns from initDF

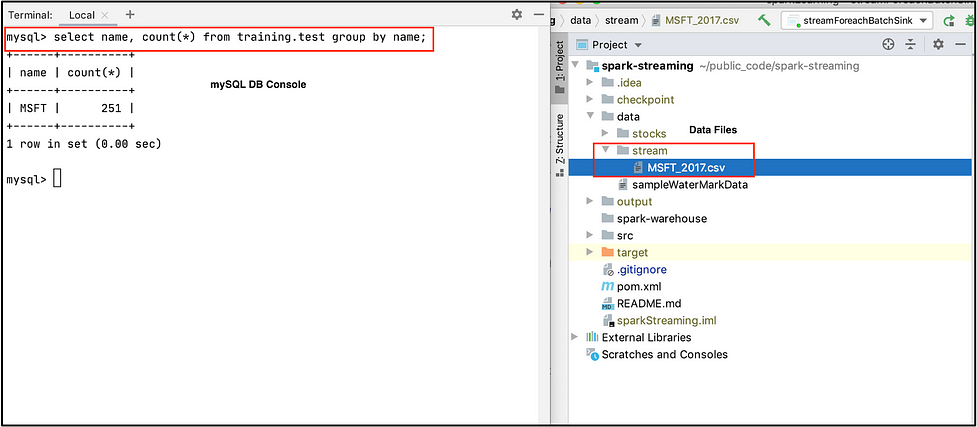
## **Output to foreachBatch sink**

foreachBatch takes a function that expects 2 parameters, first: micro-batch as DataFrame or Dataset and second: unique id for each batch. First, create a function with custom write logic to save a micro-batch to MySQL.

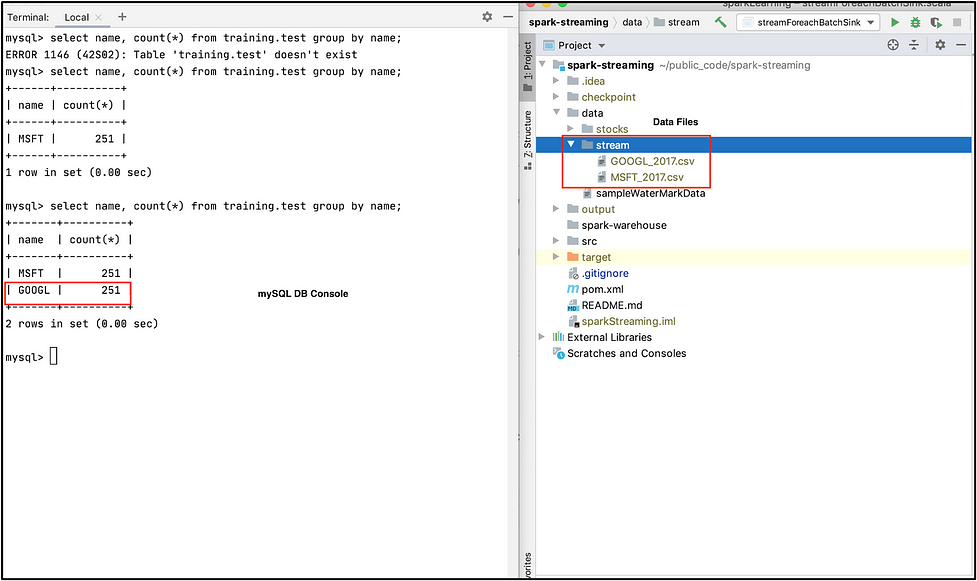
We specified all the necessary details of MySQL DB like URL, DB name (training), Table name (test), User name (root), and Password.

Use the savetoMySQL() function to save our streaming DataFrame to MySQL.

We use the foreachBatch output sink and pass the saveToMySql() function to store data in MySQL. Start the application, ingest the data, and check the result on MySQL.



In the screenshot above, we dropped the MSFT\_2017.csv file into data/stream on the right window and see the number of records for Microsoft™️ in training.test MySQL table on the left window.



Here we drop the GOOGL\_2017.csv file into data/stream on the right window and see the number of records for Google in training.test MySQL table on the left window. For easy reference, you can find the complete code on ***[GitHub](https://github.com/NeerajBhadani/spark-streaming/blob/master/src/main/scala/streamForeachBatchSink.scala" \t "https://medium.com/expedia-group-tech/_blank)***.

# **Output sink — foreach**

If you have made it this far, great! Now we discuss our last sink for this blog post. The foreach output sink performs custom write logic to each record in a streaming DataFrame. If foreachBatch is not an option, e.g. in [continuous processing](https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html" \l "continuous-processing" \t "https://medium.com/expedia-group-tech/_blank) mode or if a batch data writer does not exist, then we can use the foreach sink for custom write logic.

To use foreach we need to implement 3 methods (open, process, and close).

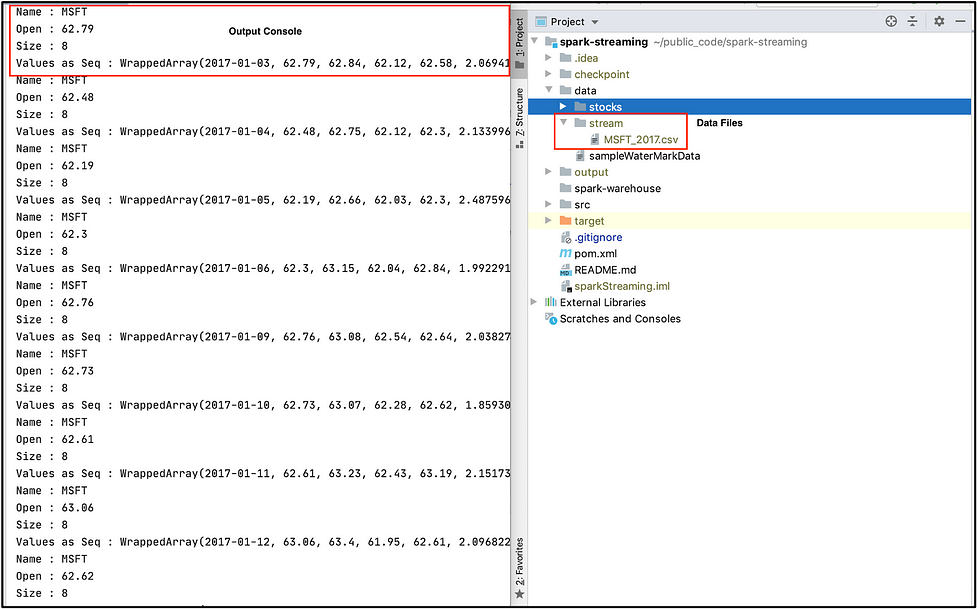
* ****open****: function to open connection
* ****process****: write data to the specified connection
* ****close****: function to close connection

## **Output to foreach sink**

We use initDF to read data from a File source. Then we create an instance of the ForeachWriter class and implement the open(), process(), and close() methods. We simply print records to console, so we use println() statements in the process() function.

Use the ForeachWriter instance defined above to write data using the foreach sink.

We used the foreach output sink and provide an instance of customWriter defined above. Execute the streaming application and check the result.



As soon as we drop the file MSFT\_2017.csv into folder data/stream on the right-hand window, we see output on the left window defined by process method in the foreach output sink. For easy reference, you can find the complete code on ***[GitHub](https://github.com/NeerajBhadani/spark-streaming/blob/master/src/main/scala/streamForeachSink.scala" \t "https://medium.com/expedia-group-tech/_blank)***.

The next post in this series covers [Checkpoints and Triggers](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-checkpoints-and-triggers-4-of-6-b6f15d5cfd8d).

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 — Input Sources](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-input-sources-2-of-6-6a72f798838c)
* [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_266

I hope you enjoyed learning about output sinks used to output data from Spark Streaming!

# **References**

## **[Structured Streaming Programming Guide](https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html" \l "output-sinks" \t "https://medium.com/expedia-group-tech/_blank)**

### [Structured Streaming is a scalable and fault-tolerant stream processing engine built on the Spark SQL engine. You can…](https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html" \l "output-sinks" \t "https://medium.com/expedia-group-tech/_blank)

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