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

# **Apache Spark Structured Streaming — Checkpoints and Triggers (4 of 6)**

## Streaming with confidence, resilience, and efficiency

[](https://medium.com/@bhadani.neeraj.08?source=post_page-----b6f15d5cfd8d--------------------------------)

[IMG_257](https://medium.com/expedia-group-tech?source=post_page-----b6f15d5cfd8d--------------------------------)

[Neeraj Bhadani](https://medium.com/@bhadani.neeraj.08?source=post_page-----b6f15d5cfd8d--------------------------------)

·

[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-checkpoints-and-triggers-4-of-6-b6f15d5cfd8d&user=Neeraj+Bhadani&userId=f9ed76fbd13a&source=post_page-f9ed76fbd13a----b6f15d5cfd8d---------------------post_header-----------)

Published in

[Expedia Group Technology](https://medium.com/expedia-group-tech?source=post_page-----b6f15d5cfd8d--------------------------------)

·

6 min read

·

Feb 25, 2021

181

2



Photo by Alexey Ruban on Unsplash

In previous blog posts, we covered using [sources](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-input-sources-2-of-6-6a72f798838c) and [sinks](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-output-sinks-3-of-6-ed3247545fbc) in Apache Spark™️ Streaming. Here we discuss ****checkpoints**** and ****triggers,**** important concepts in Spark Streaming.

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

* [Apache Spark Structured Streaming — Overview](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 — Output Sinks](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-output-sinks-3-of-6-ed3247545fbc)
* [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

# **Setup**

Let’s start creating a streaming DataFrame (resultDF) from a file source by reading a file in each micro-batch and performing aggregations. We use this resultDF for all examples in this blog post.

# **Checkpoints**

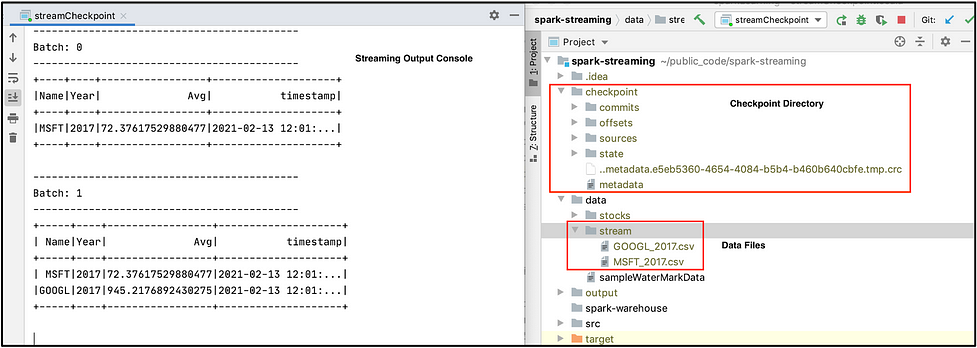
A checkpoint helps build fault-tolerant and resilient Spark applications. In Spark Structured Streaming, it maintains intermediate state on HDFS compatible file systems to recover from failures. To specify the checkpoint in a streaming query, we use the checkpointLocation parameter.

Let’s understand this with an example. We use the resultDF streaming DataFrame defined in the Setup section above.

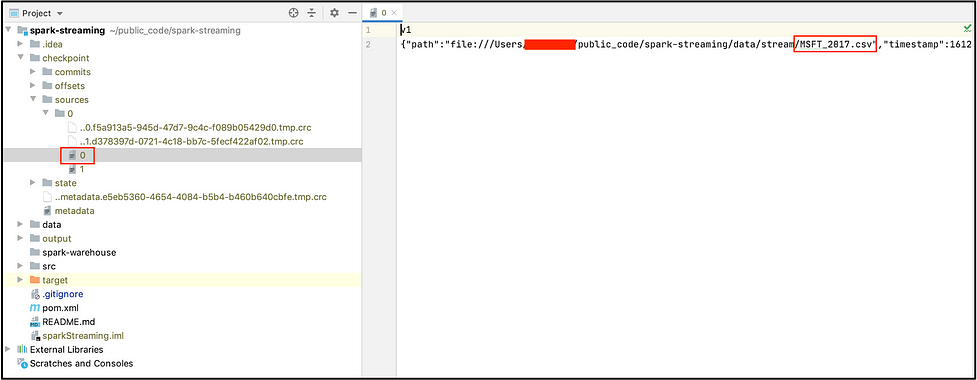
## **Output to console**

The parameter "checkpointLocation” enables the checkpoint and specifies the location where we keep checkpoint information.

Let’s execute the application and inspect the checkpoint directory.

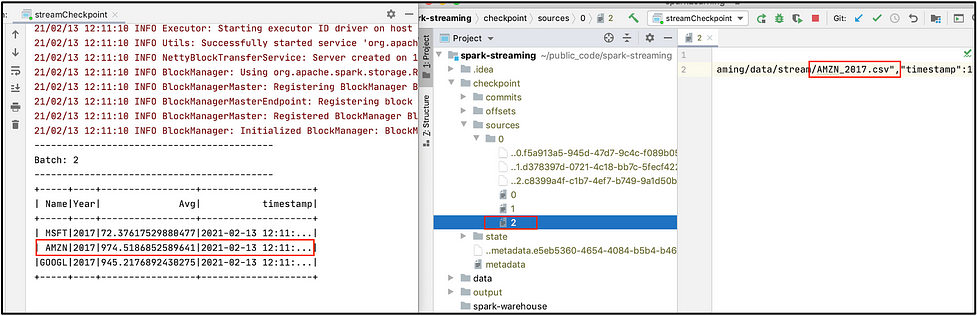


We see a directory named checkpoint with a bunch of sub-directories and files. This folder contains the state of our streaming application. E.g. the sources folder contains batches of data processed so far.



This screen shot shows the contents of the sources folder. In this example, we executed 2 batches with 1 file in each micro-batch. You can see the file MSFT\_2017.csv was the first to be processed.

Now let’s interrupt our streaming application to simulate a failure and start the application again.



Here we see an application started withBatch: 2 since it already processed Batch: 0 and Batch: 1. We also see another file, named 2, under the sources folder corresponding to AMZN\_2017.csv . This is how our application recovered from a failure. You can find the complete code on ***[GitHub](https://github.com/NeerajBhadani/spark-streaming/blob/master/src/main/scala/streamCheckpoint.scala" \t "https://medium.com/expedia-group-tech/_blank)***.

*Note: We are getting all records for MSFT, GOOGL and AMZN because we are running in complete output mode.*

# **Triggers**

By definition, data continuously flows into a streaming system. The arrival of data is not novel enough to kick off processing. In streaming systems, we need a special event to kick off processing and that event is called a trigger. Let’s discuss a few triggers in Spark Streaming.

* ****Default****: Executes a micro-batch as soon as the previous finishes.
* ****Fixed interval micro-batches****: Specifies the interval when the micro-batches will execute. For example, 1 minute , 30 seconds or 1 hour etc.
* ****One-time micro-batch****: Executes only one micro-batch to process all available data and then stops.

## **Default**

If we don’t specify any trigger, then our query will execute in micro-batch mode. The default trigger executes the next batch as soon as the previous one finishes. In our checkpoint example, we used the default trigger since we hadn't specified another.

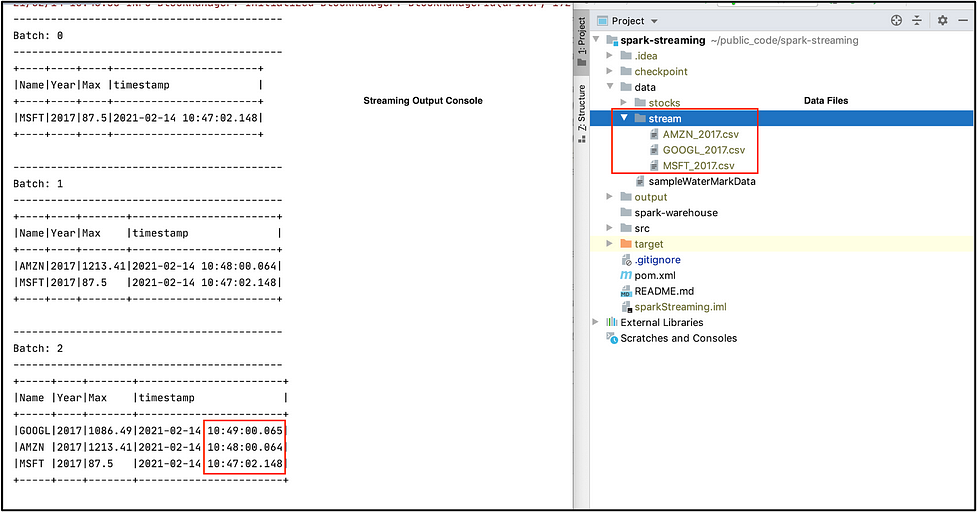
## **Fixed interval micro-batches**

Micro-batches are processed after a user-specified time interval. If the processing time of the previous batch is more than the specified interval, the next batch will be executed immediately. If we set 1 minute as the interval and a micro-batch takes 35 seconds, then the next batch will trigger after waiting for 25 seconds. If a micro-batch takes 70 seconds, then the next micro-batch execute immediately after the first ends.

Time for an example. Let’s read data from a file source, 1 file in each micro-batch, and print the result to console in complete mode. This way we can compare all the processed records. We process micro-batches every 1 minute. Again, we start with the resultDF streaming DataFrame defined above.

Here is a snippet for a fixed interval trigger set at 1 minute.

Let’s execute our streaming application!



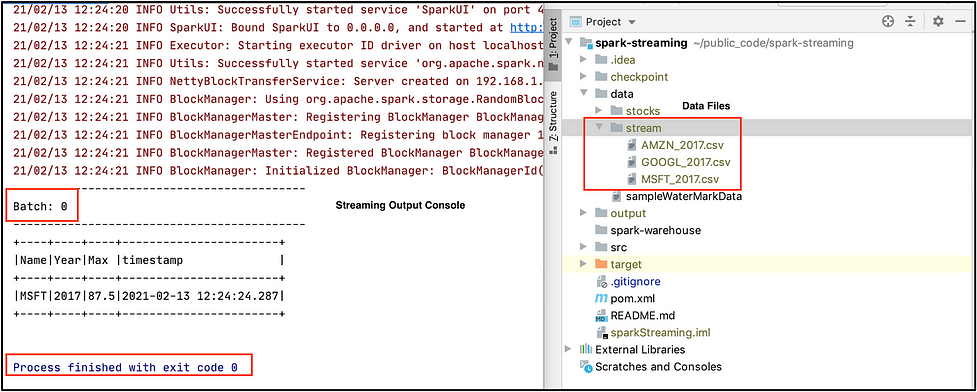
We have 3 files in the data/stream folder and specified to process 1 file in each micro-batch. Thus, we see 3 micro-batches in our output. Upon carefully inspecting the timestamp column in our output, we can see Batch: 0 was executed when we started our application and each subsequent micro-batch was executed at the start of every minute because we specified a 1 minute interval.

## **One-time micro-batch**

With a once trigger, our query will execute a single micro-batch. It will process all available data and then stop the application. This trigger is useful when you would like to spin-up a cluster periodically, to process all available data, and then shutdown the cluster. This may lead to considerable cost savings.

We reuse the set-up from the fixed interval trigger example, except we set the trigger strategy to once. We also reuse the familiar resultDF streaming DataFrame defined above in the Setup section. Here is the snippet for a trigger once.

Let’s execute our streaming application with the once trigger.



Our streaming application has only executed 1 micro-batch and completed/stopped successfully. Although we have 3 csv files in the data/stream folder, it has only processed 1 file because we specified only 1 file should be processed in each micro-batch. You can find the complete code on ***[GitHub](https://github.com/NeerajBhadani/spark-streaming/blob/master/src/main/scala/streamTriggers.scala" \t "https://medium.com/expedia-group-tech/_blank)***.

The next post in this series covers [Structured Streaming Operations](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-operations-5-of-6-40d907866fa7) like Filters, Joins, Windows, and User-Defined Functions

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 — Output Sinks](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-output-sinks-3-of-6-ed3247545fbc)
* [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 have enjoyed learning about important concepts like checkpoints and triggers in Spark Streaming.

# **Reference:**

* [http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#triggers](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html" \l "triggers" \t "https://medium.com/expedia-group-tech/_blank)
* [http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#recovering-from-failures-with-checkpointing](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html" \l "recovering-from-failures-with-checkpointing" \t "https://medium.com/expedia-group-tech/_blank)