# Spark DStream (Discretized Streams)

*Discretized Stream Operations i.e Stateless and Stateful Transformations, Output operation, Input DStream, and Receivers.*

Diagram

Description automatically generated

Spark DStream (Discretized Stream) is the basic abstraction of Spark Streaming. DStream is a continuous stream of data. It receives input from various sources like Kafka, Flume, Kinesis, or TCP sockets. It can also be a data stream generated by transforming the input stream. At its core, **DStream is a continuous stream of RDD (Spark abstraction)**. Every RDD in DStream contains data from the certain interval.

Any operation on a DStream applies to all the underlying RDDs. DStream covers all the details. It provides the developer a high-level API for convenience. As a result, Spark DStream facilitates working with streaming data.

Spark Streaming offers fault-tolerance properties for DStreams as that for RDDs. as long as a copy of the input data is available, it can recompute any state from it using the lineage of the RDDs. By default, Spark replicates data on two nodes. As a result, Spark Streaming can bear single worker failures.

## Apache Spark DStream Operations

Like RDD, Spark DStream also support two types of Operations: Transformations and output Operations-

### i. Transformation

There are two types of transformation in DStream:

* Stateless Transformations
* Stateful Transformations

#### a. Stateless Transformations

The processing of each batch has no dependency on the data of previous batches. Stateless transformations are simple RDD transformations. It applies on every batch meaning every RDD in a DStream. It includes common **RDD transformations** like map(), filter(), reduceByKey() etc.  
Although these functions seem like applying to the whole stream, *each DStream is a collection of many RDDs (batches)*. As a result, each stateless transformation applies to each RDD.

Stateless transformations are capable of combining data from many DStreams within each time step. For example, key/value DStreams have the same join-related transformations as RDDs— cogroup(), join(), leftOuterJoin() etc.

We can use these operations on DStreams to perform underlying RDD operations on each batch.

If stateless transformations are insufficient, DStreams comes with an advanced operator called transform(). transform() allow operating on the RDDs inside them. The transform() allows any arbitrary RDD-to-RDD function to act on the DStream. This function gets called on each batch of data in the stream to produce a new stream.

#### b. Stateful Transformations

It uses data or intermediate results from previous batches and computes the result of the current batch. Stateful transformations are operations on DStreams that track data across time. Thus it makes use of some data from previous batches to generate the results for a new batch.

The two main types are windowed operations, which act over a sliding window of time periods, and updateStateByKey(), which is used to track state across events for each key (e.g., to build up an object representing each user session).

Follow this link to[Read DStream Transformations in detail with the examples.](http://data-flair.training/blogs/apache-spark-streaming-transformation-operations/)

### ii. Output Operation

Once we get the data after transformation, on that data output operation are performed in Spark Streaming. After the debugging of our program, using output operation we can only save our output. Some of the output operations are print(), save() etc.. The save operation takes directory to save file into and an optional suffix. The print() takes in the first 10 elements from each batch of the DStream and prints the result.

## 4. Input DStreams and Receivers

Input DStream is a DStream representing the stream of input data from streaming source. Receiver (Scala doc, Java doc) object associated with every input DStream object. It receives the data from a source and stores it in Spark’s memory for processing.

Spark Streaming provides two categories of built-in streaming sources:

* ***Basic sources –*** These are Source which is directly available in the*StreamingContext API.* Examples: file systems, and socket connections.
* ***Advanced Sources –*** These sources are available by extra utility classes like *Kafka, Flume, Kinesis.* Thus, requires linking against extra dependencies.

For example:

* **Kafka:** the artifact required for Kafka is spark-streaming-kafka-0-8\_2.11.
* **Flume:** the artifact requires for Flume is dspark-streaming-flume\_2.11.
* **Kinesis:** the artifact required for Kinesis is spark-streaming-kinesis-asl\_2.11.

It creates many inputs DStream to receive multiple streams of data in parallel. It creates multiple receivers that receive many data stream. Spark worker/executor is a long-running task. Thus, occupies one of the cores which associate to Spark Streaming application. So, it is necessary that, Spark Streaming application has enough cores to process received data.