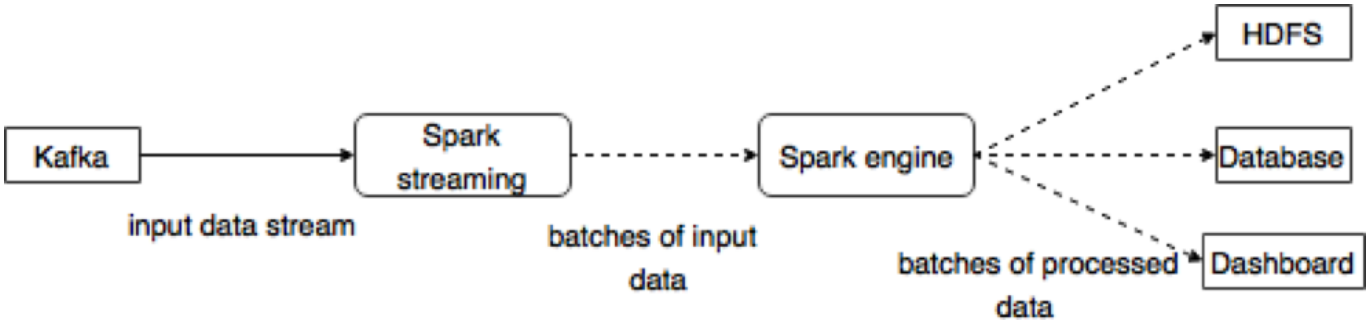
In this chapter, we will be discussing about how to integrate Apache Kafka with Spark Streaming API.

About Spark

Spark Streaming API enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Data can be ingested from many sources like Kafka, Flume, Twitter, etc., and can be processed using complex algorithms such as high-level functions like map, reduce, join and window. Finally, processed data can be pushed out to filesystems, databases, and live dash- boards. Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster.

Integration with Spark

Kafka is a potential messaging and integration platform for Spark streaming. Kafka act as the central hub for real-time streams of data and are processed using complex algorithms in Spark Streaming. Once the data is processed, Spark Streaming could be publishing results into yet another Kafka topic or store in HDFS, databases or dashboards. The following diagram depicts the conceptual flow.



Now, let us go through Kafka-Spark API's in detail.

SparkConf API

It represents configuration for a Spark application. Used to set various Spark parameters as key-value pairs.

"SparkConf" class has the following methods:

- `set(string key, string value)` - set configuration variable
- `remove(string key)` - remove key from the configuration.
- `setAppName(string name)` - set application name for your application.
- `get(string key)` - get key

StreamingContext API

This is the main entry point for Spark functionality. A SparkContext represents the connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on the cluster.

The signature is defined as shown below.

```
public  StreamingContext(String  master,
                         String  appName,
                         Duration  batchDuration,
                         String sparkHome,
                                scala.collection.Seq<String> jars,
                                scala.collection.Map<String,String> environment)
```

- `master` - cluster URL to connect to (e.g. `mesos://host:port`, `spark://host:port`, `local\[4\]`).
- `ppName` - a name for your job, to display on the cluster web UI
- `batchDuration` - the time interval at which streaming data will be divided into batches

```
public  StreamingContext(SparkConf  conf,  Duration  batchDuration)
```

Create a StreamingContext by providing the configuration necessary for a new SparkContext.

- `conf` - Spark parameters
- `batchDuration` - the time interval at which streaming data will be divided into batches

KafkaUtils API

KafkaUtils API is used to connect the Kafka cluster to Spark streaming. This API has the signifi- cant method "createStream" signature defined as below.

```
public static ReceiverInputDStream<scala.Tuple2<String,String>> createStream(
StreamingContext ssc,
              String zkQuorum,
              String groupId,
              scala.collection.immutable.Map<String,Object> topics,
              StorageLevel storageLevel)
```

The above shown method is used to Create an input stream that pulls messages from Kafka Brokers.

- `ssc` - StreamingContext object.
- `zkQuorum` - Zookeeper quorum.
- `groupId` - The group id for this consumer.
- `topics` - return a map of topics to consume.
- `storageLevel` - Storage level to use for storing the received objects.

KafkaUtils API has another method createDirectStream, which is used to create an input stream that directly pulls messages from Kafka Brokers without using any receiver. This stream can guarantee that each message from Kafka is included in transformations exactly once.

The sample application is done in Scala. To compile the application, please download and install "sbt", scala build tool (similar to maven). The main application code is presented below.

```
import java.util.HashMap
import org.apache.kafka.clients.producer.{KafkaProducer, ProducerConfig,
ProducerRecord}
import org.apache.spark.SparkConf import org.apache.spark.streaming._
import org.apache.spark.streaming.kafka._


object KafkaWordCount {
  def main(args: Array[String]) {
    if (args.length < 4) {
      System.err.println("Usage: KafkaWordCount <zkQuorum> <group>
<topics><numThreads>")
      System.exit(1)
    }

    val Array(zkQuorum, group, topics, numThreads) = args
    val sparkConf  =  new  SparkConf().setAppName("KafkaWordCount")
    val ssc  =  new  StreamingContext(sparkConf,  Seconds(2))
ssc.checkpoint("checkpoint")

    val topicMap = topics.split(",").map((_,  numThreads.toInt)).toMap
    val lines = KafkaUtils.createStream(ssc, zkQuorum, group, topicMap).map(_._2)
val words = lines.flatMap(_.split("  "))
    val wordCounts = words.map(x => (x,  1L))

    .reduceByKeyAndWindow(_ + _, _ - _, Minutes(10), Seconds(2), 2)
wordCounts.print()

    ssc.start()
    ssc.awaitTermination()

  }
}
```

Build Script

The spark-kafka integration depends on the spark, spark streaming and spark Kafka integration jar. Create a new file "build.sbt" and specify the application details and its dependency. The "sbt" will download the necessary jar while compiling and packing the application.

```
name := "Spark Kafka Project"
version := "1.0"
scalaVersion := "2.10.5"

libraryDependencies  +=  "org.apache.spark"  %%  "spark-core"  %  "1.6.0"
libraryDependencies  +=  "org.apache.spark"  %%  "spark-streaming"  % "1.6.0"
libraryDependencies  +=  "org.apache.spark"  %%  "spark-streaming-kafka"  % "1.6.0"
```

Compilation / Packaging

Run the following command to compile and package the jar file of the application. We need to submit the jar file into the spark console to run the application.

```
sbt package
```

Submiting to Spark

Start Kafka Producer CLI (explained in the previous chapter), create a new topic called "my-first-topic" and provide some sample messages as shown below.

```
Another spark test message
```

Run the following command to submit the application to spark console.

```
/usr/local/spark/bin/spark-submit  --packages  org.apache.spark:spark-streaming-
kafka_2.10:1.6.0  --class  "KafkaWordCount"  --master  local[4]
target/scala-2.10/spark- kafka-project_2.10-1.0.jar  localhost:2181  <group  name>
<topic  name>  <number  of threads>
```

The sample output of this application is shown below.

```
spark console messages ..
(Test,1)
(spark,1)
(another,1)
(message,1)
spark console message ..
```