Slide 8:
Spark streaming gives you the capability to process live streaming data in small batches. Utilizing Spark's core, Spark Streaming is scalable, high-throughput and fault-tolerant. You write Stream programs with DStreams, which is a sequence of RDDs from a stream of data. There are various data sources that Spark Streaming receives from including, Kafka, Flume, HDFS, Kinesis, or Twitter. It pushes data out to HDFS, databases, or some sort of dashboard.

Spark Streaming supports Scala, Java and Python. Python was actually introduced with Spark 1.2. Python has all the transformations that Scala and Java have with DStreams, but it can only support text data types. Support for other sources such as Kafka and Flume will be available in future releases for Python.

Slide 9:
Here's a quick view of how Spark Streaming works. First the input stream comes in to Spark Streaming. Then that data stream is broken up into batches of data that is fed into the Spark engine for processing. Once the data has been processed, it is sent out in batches.

Spark Stream support sliding window operations. In a windowed computation, every time the window slides over a source of DStream, the source RDDs that falls within the window are combined and operated upon to produce the resulting RDD.

There are two parameters for a sliding window. The window length is the duration of the window and the sliding interval is the interval in which the window operation is performed. Both of these must be in multiples of the batch interval of the source DStream.

In the diagram, the window length is 3 and the sliding interval is 2. To put it in a different perspective, say you wanted to generate word counts over last 30 seconds of data, every 10 seconds. To do this, you would apply the reduceByKeyAndWindow operation on the pairs of DStream of (Word,1) pairs over the last 30 seconds of data.

Slide 10:
Here we a have a simple example. We want to count the number of words coming in from the TCP socket. First and foremost, you must import the appropriate libraries. Then, you would create the StreamingContext object. In it, you specify to use two threads with the batch interval of 1 second. Next, you create the DStream, which is a sequence of RDDs, that listens to the TCP socket on the given hostname and port. Each record of the DStream is a line of text. You split up the lines into individual words using the flatMap function. The flatMap function is a one-to-many DStream operation that takes one line and creates a new DStream by generating multiple records (in our case, that would be the words on the line). Finally, with the words DStream, you can count the words using the map and reduce model. Then you print out the results to the console.

Slide 11:
One thing to note is that when each element of the application is executed, the real processing doesn't actually happen yet. You have to explicitly tell it to start. Once the application begins, it will continue running until the computation terminates. The code snippets that you see here is from a full sample found in the NetworkWordCount. To run the full example, you must first start up netcat, which is a small utility found in most Unix-like systems. This will act as a data source to give the application streams of data to work with. Then, on a different terminal window, run the application using the command shown here.