

Spark Streaming with Standard Streaming Scripts

Spark Streaming allows us to have application based streaming. This will help us to track statistics about pages in real time, perform real-time analytics, collect events to train machine learning models or automatically detect anomalies.

In this project, we will work with the standard streaming scripts provided by Spark on the Standalone cluster and test its efficiency.

Starting the Clusters

1. First, we need to start the Master node of the Spark Standalone Cluster.

```
libexec — -bash — 135x45
LC02RT0Y1G8WM:libexec AF34122$ sbin/start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /usr/local/Cellar/apache-spark/2.1.1/libexec/logs/spark-AF34122-org.apache.s
park.deploy.master.Master-1-lc02rt0y1g8wm.us.ad.wellpoint.com.out
LC02RT0Y1G8WM:libexec AF34122$
```

2. We can check if the Master node has started by opening a web browser and opening the address “localhost:8080”. We see the Screen showing the details of the Master Node. Keep in mind the URL of the Master Node as this will be required later.

Spark Master at spark://lc02rt0y1g8wm.us.ad.wellpoint.com:7077

URL: spark://lc02rt0y1g8wm.us.ad.wellpoint.com:7077
REST URL: spark://lc02rt0y1g8wm.us.ad.wellpoint.com:6066 (cluster mode)
Alive Workers: 0
Cores in use: 0 Total, 0 Used
Memory in use: 0.0 B Total, 0.0 B Used
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
-----------	---------	-------	-------	--------

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
----------------	------	-------	-----------------	----------------	------	-------	----------

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
----------------	------	-------	-----------------	----------------	------	-------	----------

3. Next start the Slave Node for the Cluster. Include the URL for the Master Node as a parameter.

```
libexec -- bash -- 135x45
LC02RT0Y1G8WM:libexec AF34122$ sbin/start-slave.sh spark://lc02rt0y1g8wm.us.ad.wellpoint.com:7077
starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/Celtar/apache-spark/2.1.1/libexec/logs/spark-AF34122-org.apache.s
park.deploy.worker.Worker-1-lc02rt0y1g8wm.us.ad.wellpoint.com.out
LC02RT0Y1G8WM:libexec AF34122$ clear
```

4. We can verify if the node has properly initiated in the same address.

Spark Master at spark://lc02rt0y1g8wm.us.ad.wellpoint.com:7077

URL: spark://lc02rt0y1g8wm.us.ad.wellpoint.com:7077
REST URL: spark://lc02rt0y1g8wm.us.ad.wellpoint.com:6066 (cluster mode)
Alive Workers: 1
Cores in use: 8 Total, 0 Used
Memory in use: 15.0 GB Total, 0.0 B Used
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20170512092353-30.42.221.237-49244	30.42.221.237:49244	ALIVE	8 (0 Used)	15.0 GB (0.0 B Used)

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
----------------	------	-------	-----------------	----------------	------	-------	----------

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
----------------	------	-------	-----------------	----------------	------	-------	----------

Starting the Streaming Program

1. Run netcap command on Port 9999. I am using Mac OS the command may vary with the OS. This command basically allows us to send data to a specific port.

```
libexec -- nc -lk 9999 -- 163x51
LC02RT0Y1G8WM:libexec AF34122$ nc -lk 9999
```

2. On a different terminal, from the run-example folder of Spark, start the 'streaming.NetworkWordCount' with the local port of 9999, the same port where we are running netcap. You will see timestamps after every second on the second, the timestamps being in the format of seconds since the Unix epoch in 1970.

```

libexec — java -cp /usr/local/Cellar/apache-spark/2.1.1/libexec/conf:/usr/local/Cellar/apache...
LC02RT0Y1G8WM:libexec AF34122$ bin/run-example streaming.NetworkWordCount localhost 9999
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
17/05/12 09:38:22 INFO StreamingExamples: Setting log level to [WARN] for streaming example
. To override add a custom log4j.properties to the classpath.
17/05/12 09:38:22 WARN NativeCodeLoader: Unable to load native-hadoop library for your plat
form... using builtin-java classes where applicable

-----
Time: 1494596305000 ms
-----

Time: 1494596306000 ms
-----

Time: 1494596307000 ms
-----

Time: 1494596308000 ms
-----

Time: 1494596309000 ms
-----

Time: 1494596310000 ms
-----

```

Testing the Streaming

1. Keeping both the windows side-by-side, start entering a few messages on the netcap command window. First we will enter a few words on a line. We will notice that every time we enter the data and press enter, Spark not only gathers the data in that second, it also reduces it into groups with word count.

```

Terminal Shell Edit View Window Help
libexec — nc -lk 9999 — 105x48
LC02RT0Y1G8WM:libexec AF34122$ nc -lk 9999
hello
hello
hello hello
Aditya Gogoi Gogoi Aditya

libexec — java -cp /usr/local/Cellar/apache-spark/2.1.1/libexec/conf:/usr/local/Cellar/apache...
Time: 1494596345000 ms
-----
Time: 1494596346000 ms
-----
Time: 1494596347000 ms
-----
Time: 1494596348000 ms
-----
Time: 1494596349000 ms
-----
Time: 1494596350000 ms
-----
Time: 1494596351000 ms
-----
(Gogoi,2)
(Aditya,2)
-----
Time: 1494596352000 ms
-----
Time: 1494596353000 ms
-----
Time: 1494596354000 ms
-----
Time: 1494596355000 ms
-----

```

2. Next we will just make another try with a coherent sentence. We see that Spark Streaming easily parses the sentence into its component words and reduces it to its wordcount.

```
libexec -- nc -lk 9999 -- 105x48
LC02RT0Y1G8WM:libexec AF34122$ nc -lk 9999
heloo
heloo
halo halo
Aditya Gogoi Gogoi Aditya
How are you ? are you OK ?

libexec -- java -cp /usr/local/Cellar/apache-spark/2.1.1/libexec/conf:/usr/local/Cellar/apache...
-----
Time: 1494596387000 ms
-----
Time: 1494596388000 ms
-----
Time: 1494596389000 ms
-----
Time: 1494596390000 ms
-----
Time: 1494596391000 ms
(How,1)
(are,2)
(OK,1)
(you,2)
(? ,2)
-----
Time: 1494596392000 ms
-----
Time: 1494596393000 ms
-----
Time: 1494596394000 ms
-----
Time: 1494596395000 ms
-----
Time: 1494596396000 ms
```

We can see that Spark Streaming is a powerful tool which provides streaming capabilities on any application or network. This proves useful when we have a constant stream of data and we need insights on it on a regular time interval.