**Developers Guide to**

**Spark2**

Version -V1.0

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Description | Author |
| 03/15/2018 | 1.0 | Initial Version | Yagnaraman Madhavan |
|  |  |  |  |
|  |  |  |  |

* **Version History**

Contents

[Spark Batch Application 4](#_Toc511153104)

[1.0 Logging into the REPL 4](#_Toc511153105)

[2.0 Interactive Data Processing: REPL 5](#_Toc511153106)

[3.0 Executing Spark Batch application using Spark submit 5](#_Toc511153107)

[4.0 Executing hive queries using Spark SQL & JDBC connection in Beeline: 11](#_Toc511153108)

[5.0 Talend Spark Batch Job 11](#_Toc511153109)

[Spark Streaming 14](#_Toc511153110)

[6.0 Kafka Consumer – Spark Streaming Integration: Structured streaming 14](#_Toc511153111)

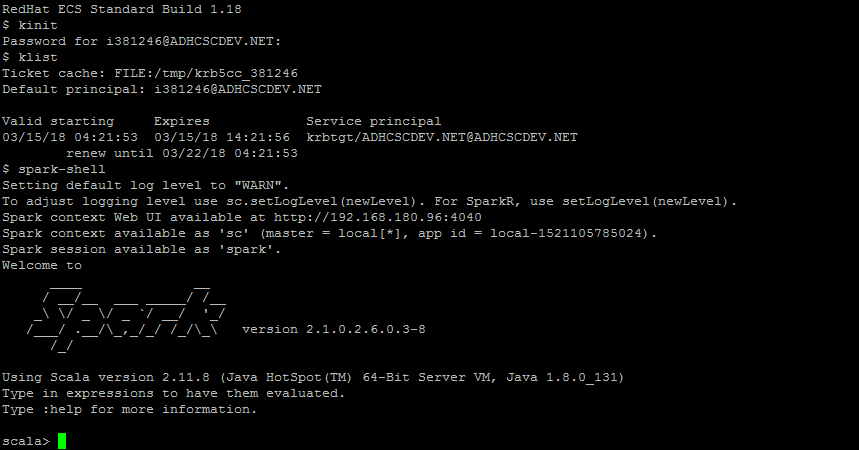
[7.0 Best Practices: Spark Development 15](#_Toc511153112)

[8.0 Scala Programming Style Guide: 16](#_Toc511153113)

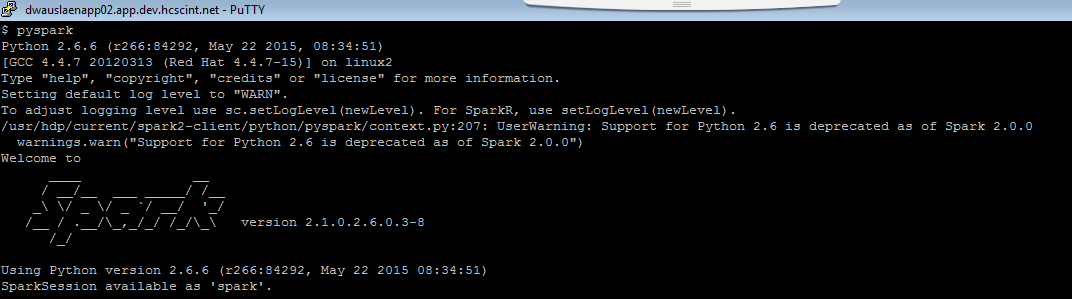
# Spark Batch Application

# Logging into the REPL

* **Scala shell**

****

* Type :q or sys.exit to exit from the shell
* By default, spark context & spark session will be available.
* **Python shell**

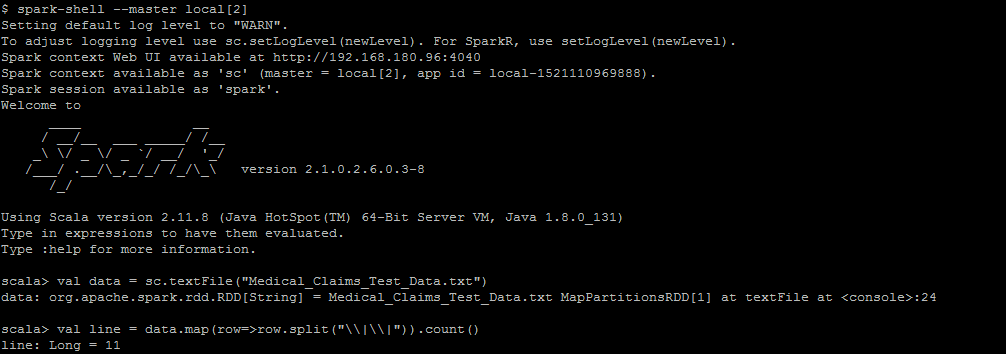
****

* By default, spark context & spark session will be available.
* Type exit() to exit from REPL.

# Interactive Data Processing: REPL

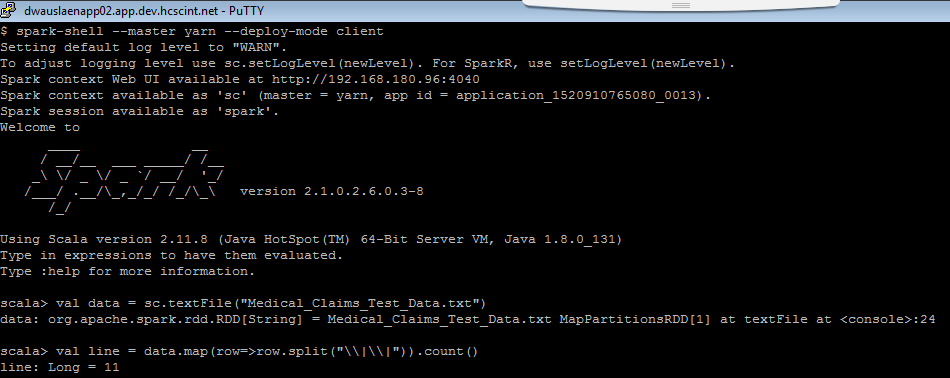
Spark shell in Local Mode

* Spark shell with deployment mode => Local 2 threads (cores)

******

Spark shell with Yarn client mode

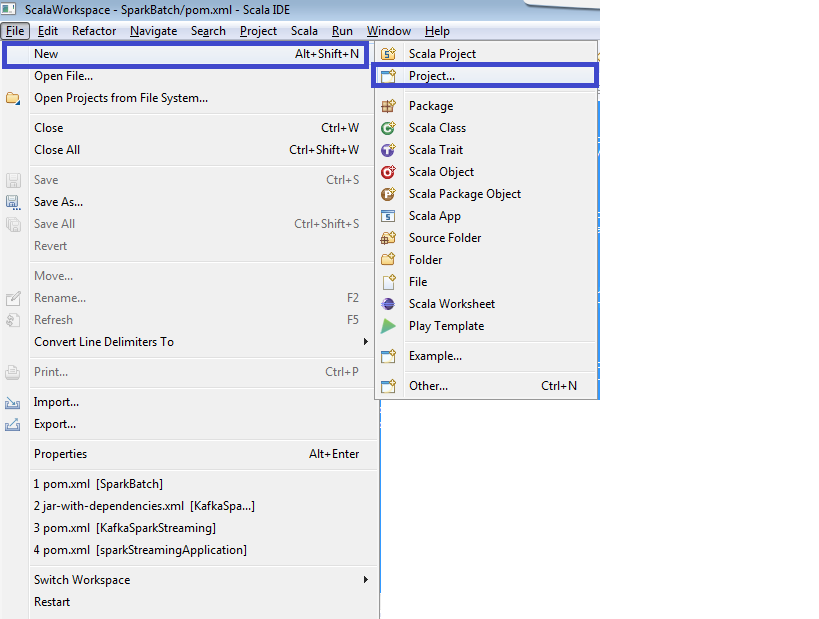
* Spark shell with deployment mode => Yarn client

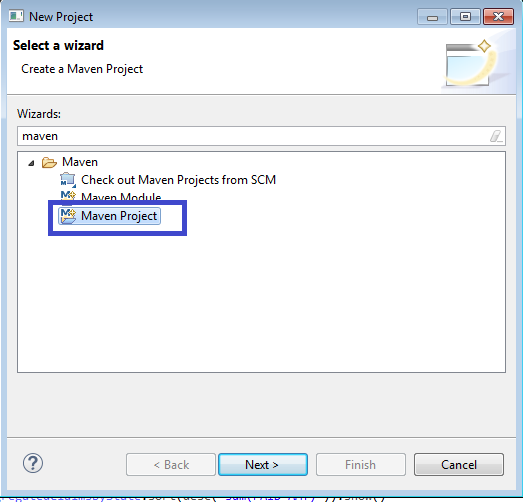
******

# Executing Spark Batch application using Spark submit

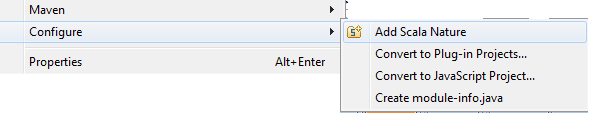
***Setting up Maven in Eclipse: for Scala***

1. While setting up maven for eclipse, make sure that scala and spark versions are compatible with each other.
   1. Refer versions compatibility in maven website to ensure that there is compatibility between scala & spark.
2. Create a maven project and add scala nature to the project.

**

**

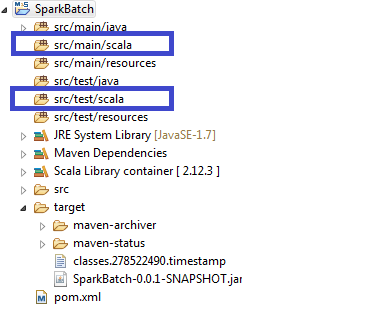
After the maven project is created, add scala nature.

**

1. Ensure that all dependencies like scala-maven plugin are given in pom.xml
   1. Refer the plugins section in pom.xml
2. Ensure that all required libraries like spark core, spark sql, scala libraries, etc.. are also given in pom.xml
   1. Refer the dependencies section in pom.xml file

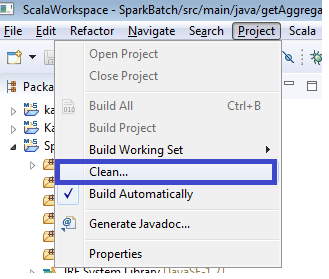


1. Add the required folders like src/main/scala, src/test/scala.

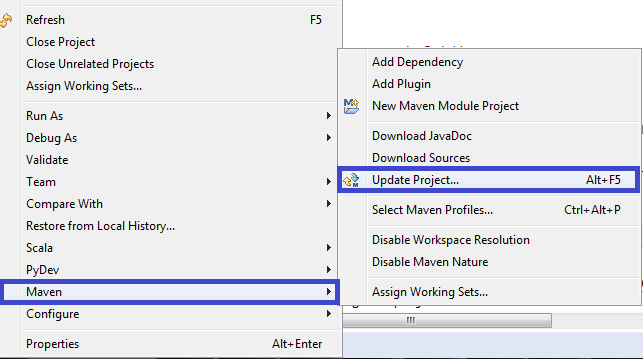
******

Package & Bundle the jar:

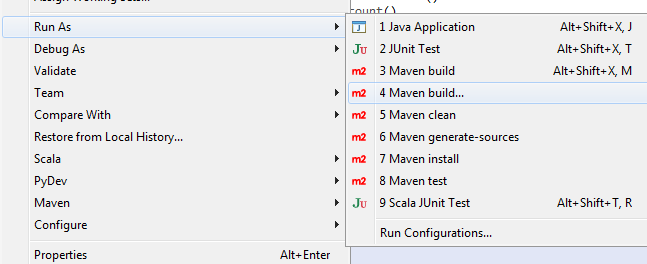
1. Clean & select your project folder to build from scratch

******

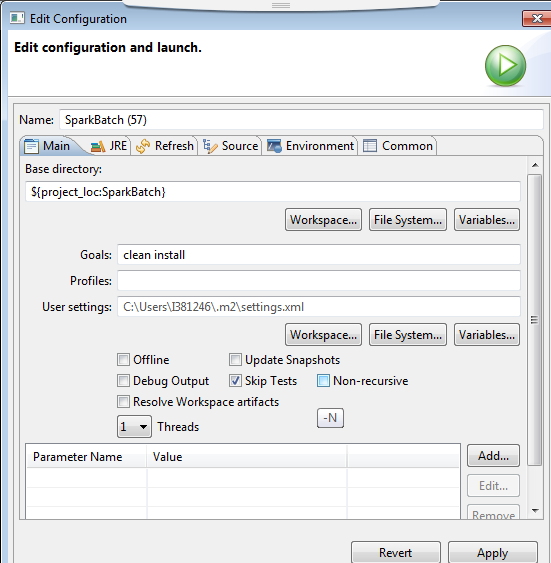
1. Right click on your project folder 🡪 Click on Maven -> Update project

******

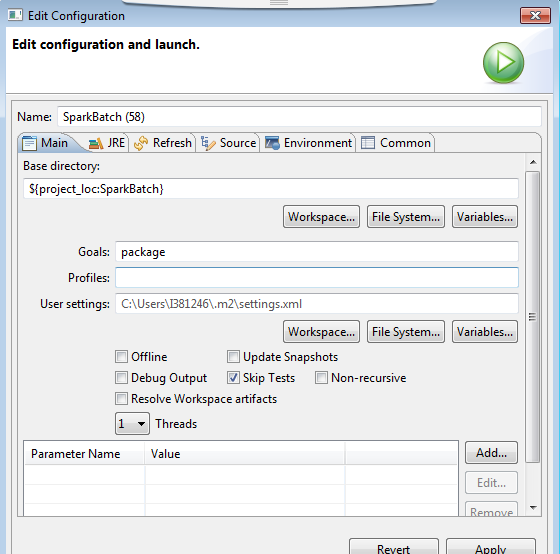
1. Maven clean install
   1. Right click on your project folder-> select Run as -> select Maven build

******

* 1. Clean install skip test to skip
     1. Clean : removes all files generated by previous builds
     2. Install: installs the package in the local repository
     3. Skip Tests : skip unit tests

******

* 1. Package
     1. Package : packages the compiled code in the distribution format

******

Executing spark batch applications through spark-submit:

Local Mode :

spark-submit --class com.hcsc.spark.getAggregatedClaimsByState --master=local[2] SparkBatch-0.0.1-SNAPSHOT.jar Medical\_Claims \_Test \_Data.txt

Yarn Client Mode:

spark-submit --class com.hcsc.spark.getAggregatedClaimsByState --master=yarn --deploy-mode=client SparkBatch-0.0.1-SNAPSHOT.jar Medical\_Claims \_Test \_Data.txt

Yarn ClusterMode:

spark-submit --class com.hcsc.spark.getAggregatedClaimsByState --master=yarn --deploy-mode=cluster SparkBatch-0.0.1-SNAPSHOT.jar Medical\_Claims \_Test \_Data.txt

The following configuration options can also be given as part of spark-submit mode:

--num-executors: To configure how many executors will be allocated

--executor-memory: RAM per executors

--executor-cores: CPU cores per executor

# Executing hive queries using Spark SQL & JDBC connection in Beeline:

Thriftserver details: Dev On-prem

dwauslenapp02.app.dev.hcscint.net:10016

Hive Queries can be executed using JDBC connection & sparksql & thriftserver

There are 2 methods to execute hive queries using spark:

1. HTTP method
2. Non-HTTP method.

beeline -u "jdbc:hive2://dwauslenapp02.app.dev.hcscint.net:10016/;transportmode=http;principal=hive/\_HOST@ADHCSCDEV.NET;httpPath=cliservice"?

**The following connection string accesses Spark SQL through JDBC on a Kerberos-enabled cluster:**

beeline -u "jdbc:hive2://dwauslenapp02.app.dev.hcscint.net:10016/;transportmode=http;principal=hive/\_HOST@ADHCSCDEV.NET;httpPath=/"?

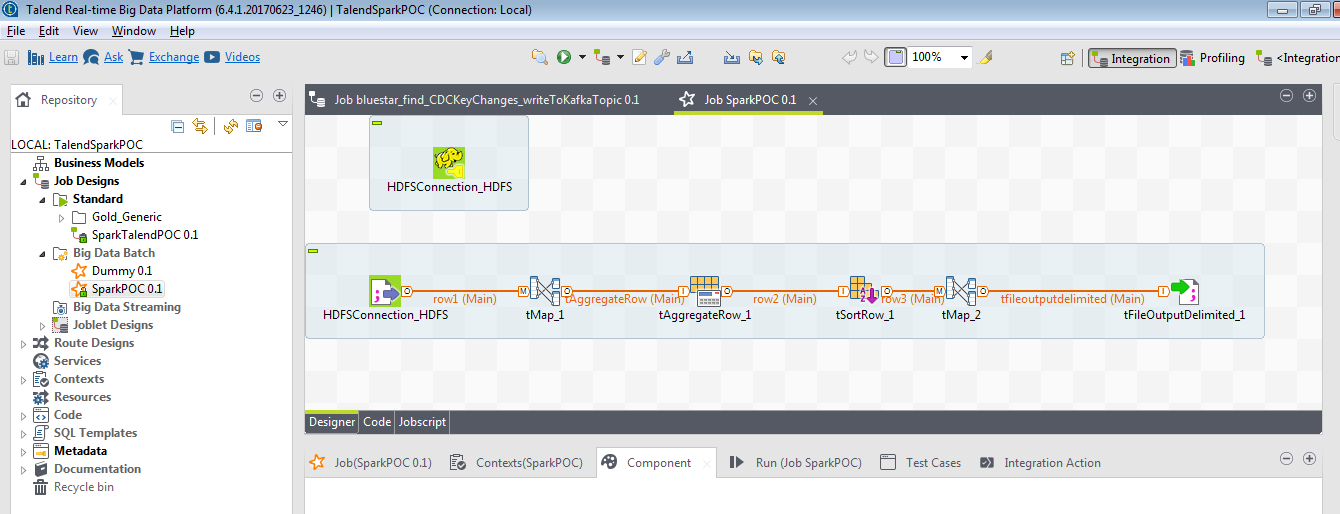
**The following connection string accesses Spark SQL through JDBC over HTTP transport on a Kerberos-enabled cluster:**

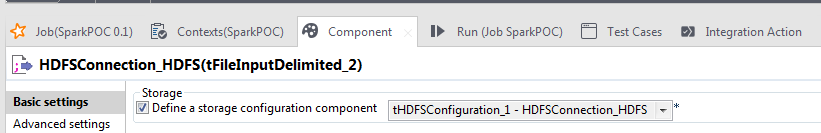
beeline -u "jdbc:hive2://dwauslenapp02.app.dev.hcscint.net:10016/;principal=hive/\_HOST@ADHCSCDEV.NET;httpPath=/"?

# Talend Spark Batch Job

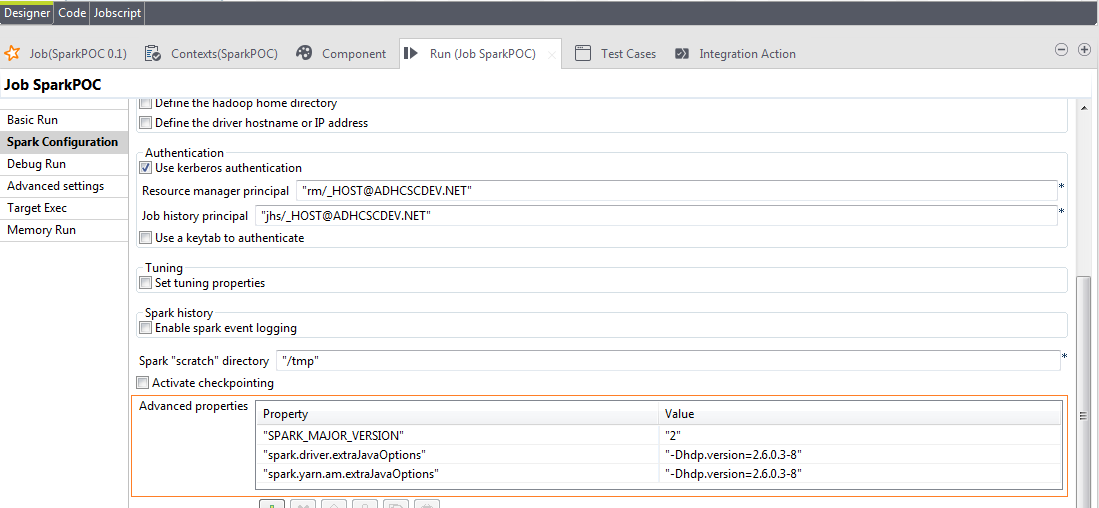
Talend spark batch job needs to be created as a big data batch job in talend.

Create HDFS/Hive/Hbase/Kafka etc.. Connection and configure it in define a storage configuration component.

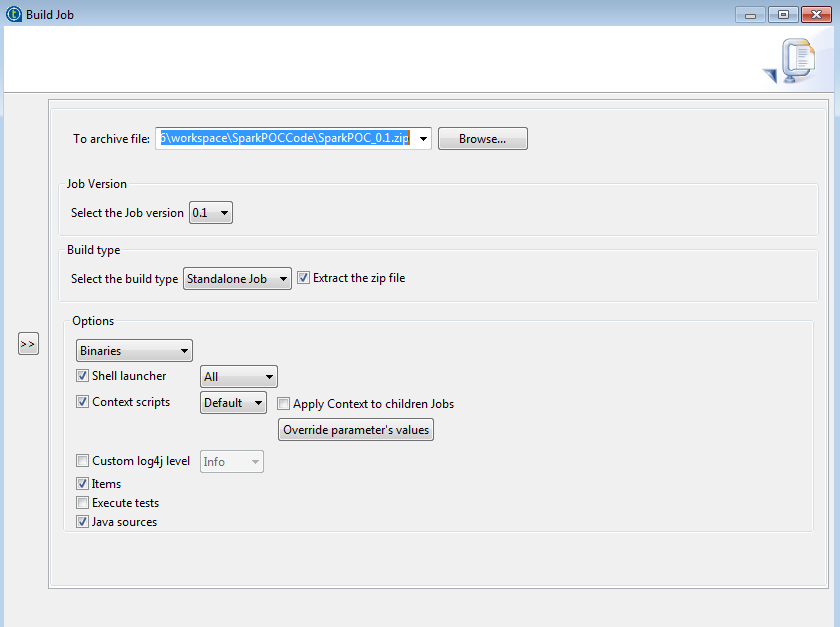




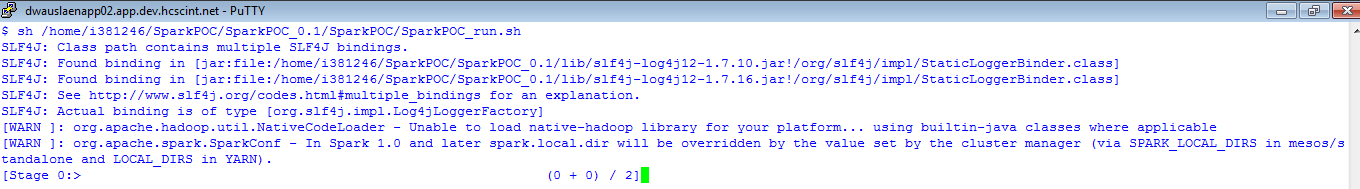
Configure the below advanced properties when running the job in local using spark execution engine.



Build the Talend batch job:



Extract the zip file contents and move the extracted files to edge node. Execute the shell script.



# Spark Streaming

**Guidelines:**

Window Interval & sliding interval should be set as the multiple of the batch interval and sliding interval should never be set greater than window interval.

**BatchInterval** defines how frequently spark streaming should collect data (say 2-4 seconds). If spark streaming can process data much faster, then reduce the batch interval to process the data faster.

If windowed computations are needed, set the below parameters:

**Window Interval:** This is the duration of the window for which computation needs to be performed.

**Sliding Interval:** This is how frequently operation needs to be performed within the window.

# Kafka Consumer – Spark Streaming Integration: Structured streaming

1. Create a context with a defined batch interval.
2. To consume data from kafka, kafka broker servers (Bootstrap servers), Topics and consumer group ID are required.
3. Use structured streaming to utilize spark SQL execution engine using spark.readstream.schema(schema).format().load() and perform ETL on the streams of data.[Structured Streaming is essentially a stream-processing engine built on top of the Spark SQL engine]
4. Specify startingOffsets to be “earliest”, which will read all data available in the topic at the start of the query.
5. Parse the dataframe, do the computations and write the data to the sink (Kafka, HDFS, Hbase etc..)
6. Ensure that data is checkpointed in a HDFS location to ensure recovery & fault tolerance.
7. Start the spark streaming.
8. Await spark streaming termination.

**Note:** If the startingOffsets option is not specified, the default value of “latest” is used and only data that arrives after the query starts will be processed.

***Setting up Maven in Eclipse: for Scala***

1. While setting up maven for eclipse, make sure that scala and spark versions are compatible with each other.
2. Create a maven project and add scala nature to the project.
3. Ensure that all dependencies like scala-maven plugin are given in pom.xml
4. Ensure that all required libraries like spark sql, scala libraries,Spark Streaming etc.. are also given in pom.xml
5. Add the required folders like src/main/scala, src/test/scala.

**Sample pom.xml - streaming application:**

****

# Best Practices: Spark Development

To improve spark performance, assess and tune the following operations:

1. Minimize shuffle operations where possible.
2. Match join strategy (ShuffledHash join vs BroadcastHash join) to the table.
3. Shufflehash join is the default join and works wellwhen data is not skewed and evenly distributed among the keys. Broadcast Hash join is used when one of the datasets is small enough to fit in memory.
4. Use kryo serializer, snappy compression & enable tungsten to improve performance.
5. Use dataframes/datasets where possible. As progressive versions of spark roll out, there is a lot of performance optimization of dataframes/datasets operations that won't be benefited from with just rdd operations.
6. If too many high-complexity or resource intensive steps are being taken between execution, it puts a lot of strain on the system and the resources available.

If there is a lot of wide transformations going on, streamline the workload to either decrease the number of said transformations or else make them more efficient.

1. When a dataset no longer needs to be in memory remember to unpersist it. At the end of a job, make sure to unpersist any sets that were persisted to make sure there are no artifacts left over. The commands that can be used for this situation are: coalesce() and repartition().
2. Only persist datasets when necessary. This will hold the entire dataset in memory, which, while speeding up work, will also chew up available RAM for complex operations.
3. If streaming jobs are being used, make sure to take advantage of checkpointing. It adds an extra layer of reliability, as spark streaming is a really just microbatching.
4. Minimize the use of <operation>ByKey (other than reduceByKey, which is recommended). These functions have a lot of overhead shuffling data via the internal hashtables.
5. Don't always rely on the default rdd partitioning. It will generally always split things into 100 pieces unless otherwise defined.
6. To effectively debug spark RDD issues & to identify the dependencies, use <rddname>.toDebugString().
7. To get the query plan when using the dataframe API, use <DataFrameName>.explain()
8. Use inferschema, True option if the file already has a header and if file doesn’t have a header, then use case class to define the header , datatypes and use it while creating the spark dataframe.

# Scala Programming Style Guide:

****