**Building a package and submit as an application**

# Right click on the project  Run As  Maven Clean Right click on the project  Run As  Maven Install

**Goto Linux terminal**

cd ~

**Submit Jar to different clusters:-**

**Local Mode**

* It means the JVM container in which the driver programs runs is created locally in this machine.
* It is used for development purpose.
* When productionizing the code we need to mention some to her mode.

spark-submit --class org.inceptez.streaming.filestream --master local[2]

/home/hduser/workspacespark/spark/target/spark-0.0.1-SNAPSHOT-jar-with-dependencies.jar

**Standalone Mode**

* This is similar to Local mode. But the only difference is, it can be ran in any data node which is inside any cluster.
* Its not restricted to the current local machine.

start-master.sh start-slaves.sh

**Submit the jar in standalone mode**

spark-submit --class org.inceptez.streaming.filestream --master spark://localhost:7077

/home/hduser/workspacespark/spark/target/spark-0.0.1-SNAPSHOT-jar-with-dependencies.jar

**Yarn mode**

* This is the case we use in real time production.
* This means the spark job will be submitted to the YARN framework which is one of the processing engine used in a cluster.

spark-submit --class org.inceptez.streaming.kafkastream --master yarn

/home/hduser/workspacespark/spark/target/spark-0.0.1-SNAPSHOT-jar-with-dependencies.jar

spark-submit --class org.inceptez.streaming.filestream --master yarn

/home/hduser/workspacespark/spark/target/spark-0.0.1-SNAPSHOT.jar --driver-memory 512m --num-executors 1 --executor-cores 2 --executor-memory 512m

spark-submit --class org.inceptez.stream.filestream --master yarn

/home/hduser/install/Sparktwitterusecase/target/sparkIncetez-0.0.1-SNAPSHOT.jar --driver- memory 512m --num-executors 1 --executor-cores 1 --executor-memory 1 --spark-shuffle- compress true --spark-speculation true --spark-dynamicAllocation-enabled true --spark- dynamicAllocation-minExecutors 1 --spark-dynamicAllocation-maxExecutors 1 --spark-

dynamicAllocation-initialExecutors 1

spark-submit --verbose \

--class org.inceptez.stream.filestream \

--master yarn /home/hduser/install/Sparktwitterusecase/target/sparkIncetez-0.0.1- SNAPSHOT.jar \

--deploy-mode client \

--queue default \

--driver-memory 512m \

--num-executors 1 \

--executor-cores 2 \

--executor-memory 512m \

--spark-shuffle-compress true \

--spark-speculation true \

--spark-dynamicAllocation-enabled true \

--spark-dynamicAllocation-initialExecutors 1 \

--spark-dynamicAllocation-minExecutors 1 \

--spark-dynamicAllocation-maxExecutors 8 \

--spark-dynamicAllocation-executorIdleTimeout 30s \

--conf spark.shuffle.service.enabled true \

--conf spark.sql.shuffle.partitions=4

[**https://spark.apache.org/docs/2.0.1/configuration.html**](https://spark.apache.org/docs/2.0.1/configuration.html)

**Lets see in detail above the above mentioned parameters while submitting a Spark job.**

--verbose \

🡪 this displays additional information on the log.

--class org.inceptez.stream.filestream \

🡪 this is the program which we are going to run

--master yarn /home/hduser/install/Sparktwitterusecase/target/sparkIncetez-0.0.1- SNAPSHOT.jar \

🡪 This specifies which type of cluster/processing engine we use to execute the job.

If we mention YARN, this job will be taken care by YARN framework.

--deploy-mode client \

🡪 this specifies where the spark job should be deployed.

- if client, then JVM will be created in the client node and the driver program (pgm which has main method) will be loaded inside that JVM.

- usually client mode is not preferred, because client node’s memory will be accessed by many other edge node applications and it may be a overhead.

- If cluster, then JVM will be created in any node in the cluster and Driver will be loaded there.

--queue default \

🡪 there are lot of queuing concepts such as FIFO, etc..

Each cluster has customized queue set up by admin team which will has different weightage in priority.

--driver-memory 512m \

🡪 **Driver** in Spark = **App** **Master** in Yarn

**Executer** in Spark = **containers** in Yarn.

This is the memory size of the JVM container in which driver program is loaded.

--num-executors 1 \

🡪 The above parameter specifies how many executors need to be used for this job.

But while determining this number we need to consider about capacity required by other users and many other criteria which we will discuss later.

--executor-cores 2 \

🡪 Mentions how many core needed for this job.

That many parallel threads will be created to compute this job in parallel.

--executor-memory 512m \

🡪 this is the memory size for executor in which the corresponding task will be performed.

--spark-shuffle-compress true \

🡪 specifies whether compression should be done during shuffling.

Shuffling means data transfer between nodes. Typically during a process in getting executed.

For example, while stage shift shuffling happens..

|  |  |
| --- | --- |
| **Just a copy** | **shuffling** |
| If data is copied from node A to node B. | If data is copied from  A1 to B1,B2,B3  A2 to B1,B3  A3 to B1,B2 |

--spark-speculation true \

🡪 Speculation means, when a task is running for a long time, then another duplicate task will be initiated in parallel.

Which even task finishes first, its result will be considered and the other task will be killed.

--spark-dynamicAllocation-enabled true \

🡪 we can mention the no of executer, min and max limit..

If dynamic allocation is enabled, then spark will take care of initiating required executer with in the min,max range dynamically for efficiency.

--spark-dynamicAllocation-initialExecutors 1 \

🡪 when dynamic allocation is enabled, this specified the initial number of executors to be created.

--spark-dynamicAllocation-minExecutors 1 \

🡪 when dynamic allocation is enabled, this specified the minimum number of executors to be created.

--spark-dynamicAllocation-maxExecutors 8 \

🡪 when dynamic allocation is enabled, this specified the maximum number of executors to be allowed.

--spark-dynamicAllocation-executorIdleTimeout 30s \

🡪 When no tasks are to be executed, the executor becomes idle. By default, 60 seconds of idle executor will be removed. This value can be controlled through the key

--conf spark.shuffle.service.enabled true \

🡪 Enables the external shuffle service. This service preserves the shuffle files written by executors so the executors can be safely removed. This must be enabled if spark.dynamicAllocation.enabled is "true". The external shuffle service must be set up in order to enable it

--conf spark.sql.shuffle.partitions=4

🡪 configures the number of partitions that are used when shuffling data for joins or aggregations.

To Start spark shell with yarn-client

spark-shell --verbose --master yarn-client --driver-memory 512m --num-executors 1 -- executorcores 2 --executor-memory 512m

or

spark-shell --master yarn

Spark Memory Allocation

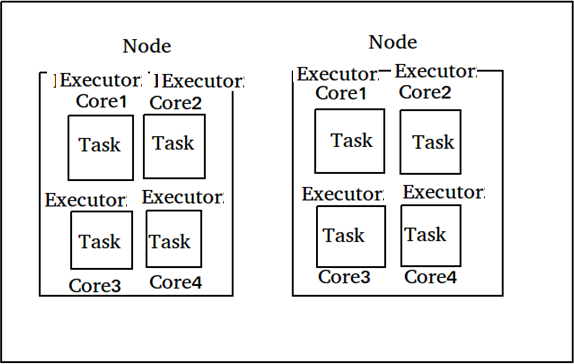
Let’s consider a 10 node cluster with following config and analyse different possibilities of executors- core-memory distribution:

\*\*Cluster Config:\*\*

12 node ( 10 Worker/Data Nodes, 1 Master, 1 Client) 16 cores per Node

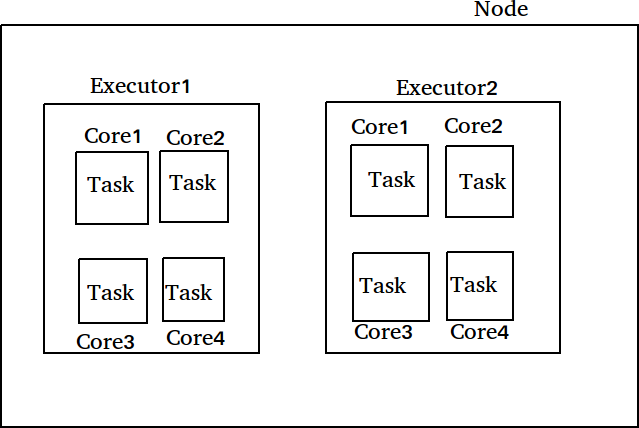
64GB RAM per Node

First Approach: Tiny executors [One Executor per core]:

1. **Multiple tasks in the same executor JVM can't be run, huge number of small JVMs start and kill overhead.**
2. **Shared/cached variables like broadcast variables and accumulators will be replicated in each executor of the nodes which is 16 times.**
3. **Not leaving enough memory overhead for Hadoop/Yarn daemon processes and not counting in ApplicationManager.**

Second Approach: Fat executors (One Executor per node)

1. **With all 16 cores per executor,**
2. **ApplicationMaster and daemon processes are not counted**
3. **HDFS throughput will hurt and it’ll result in excessive garbage results.**



Third Approach: Balance between Fat (vs) Tiny

With the cluster of 10 Nodes , 16 cores per Node, 64GB RAM per Node

Assign 5 cores per executors => --executor-cores = 5 (for good HDFS throughput)

Leave 1 core per node for DN/Yarn daemons => Num cores available per node = 16-1 = 15 Total available of cores in cluster = 15 cores x 10 nodes = 150 cores

Number of available executors = (total cores/num-cores-per-executor) = 150 total cores/5 executor cores = 30 executors available

Leave 1 executor for ApplicationMaster => 30 – 1 = 29 total executors

Number of executors per node => --num-executors = 30 executors / 10 nodes= 3 executors per node Memory per executor => 64GB pernode /3 executors = 21GB per executor

Memory per executor after Counting off heap overhead = off heap allocation of ~7% of 21GB = 3GB, hence --executor-memory = 21 - 3 = 18GB Memory per executor

So, recommended config is: 29 executors, 18GB memory each and 5 cores each!! Third approach has found right balance between Fat vs Tiny approaches.

Finally it achieved parallelism of a fat executor and best throughputs of a tiny executor!!

Leaving all the above calculations, iterate or benchmark for optimal/better performances considering ondemand/continuous other resources consumption in the cluster for other jobs at that time.