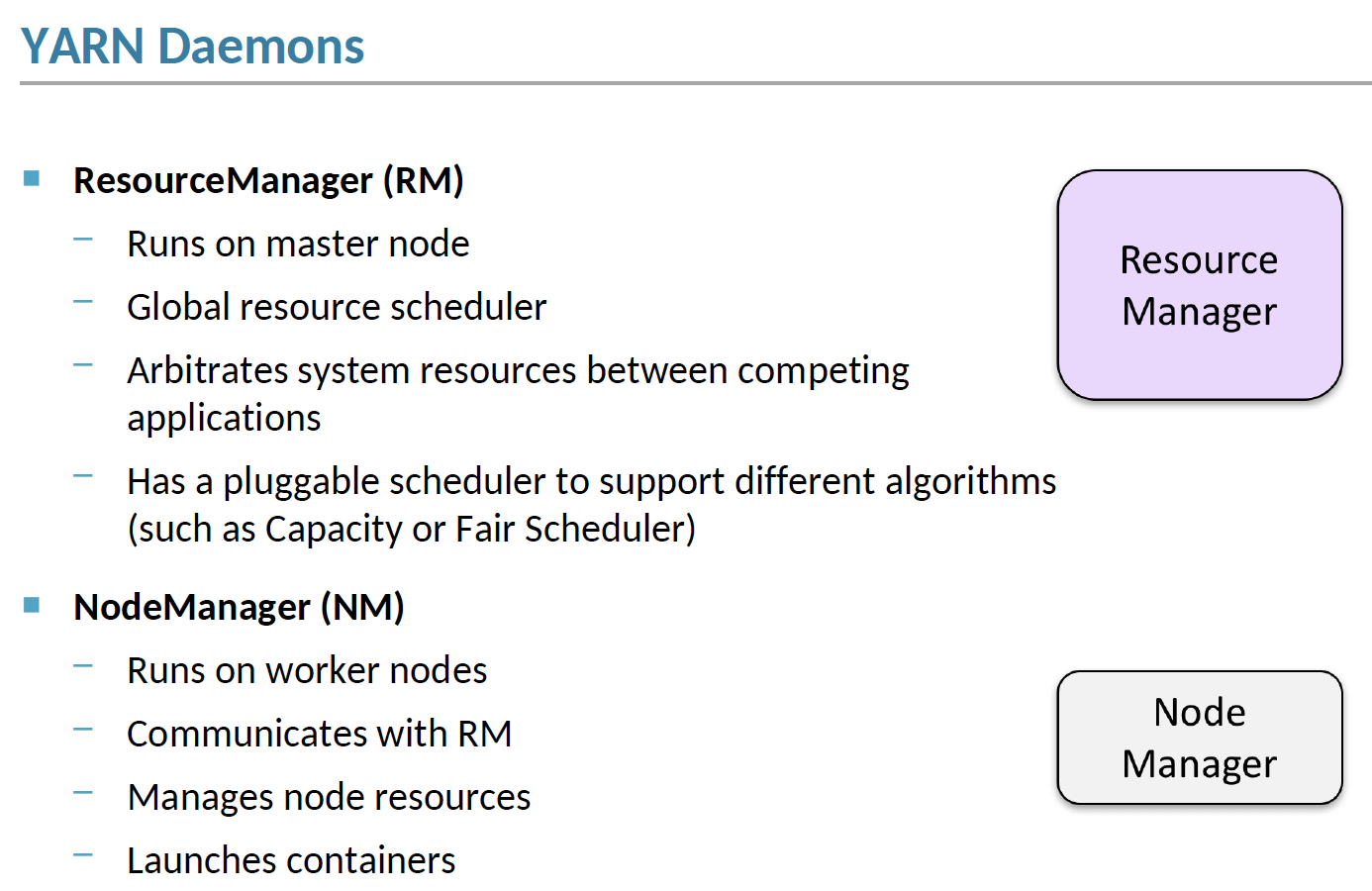
What Is YARN?

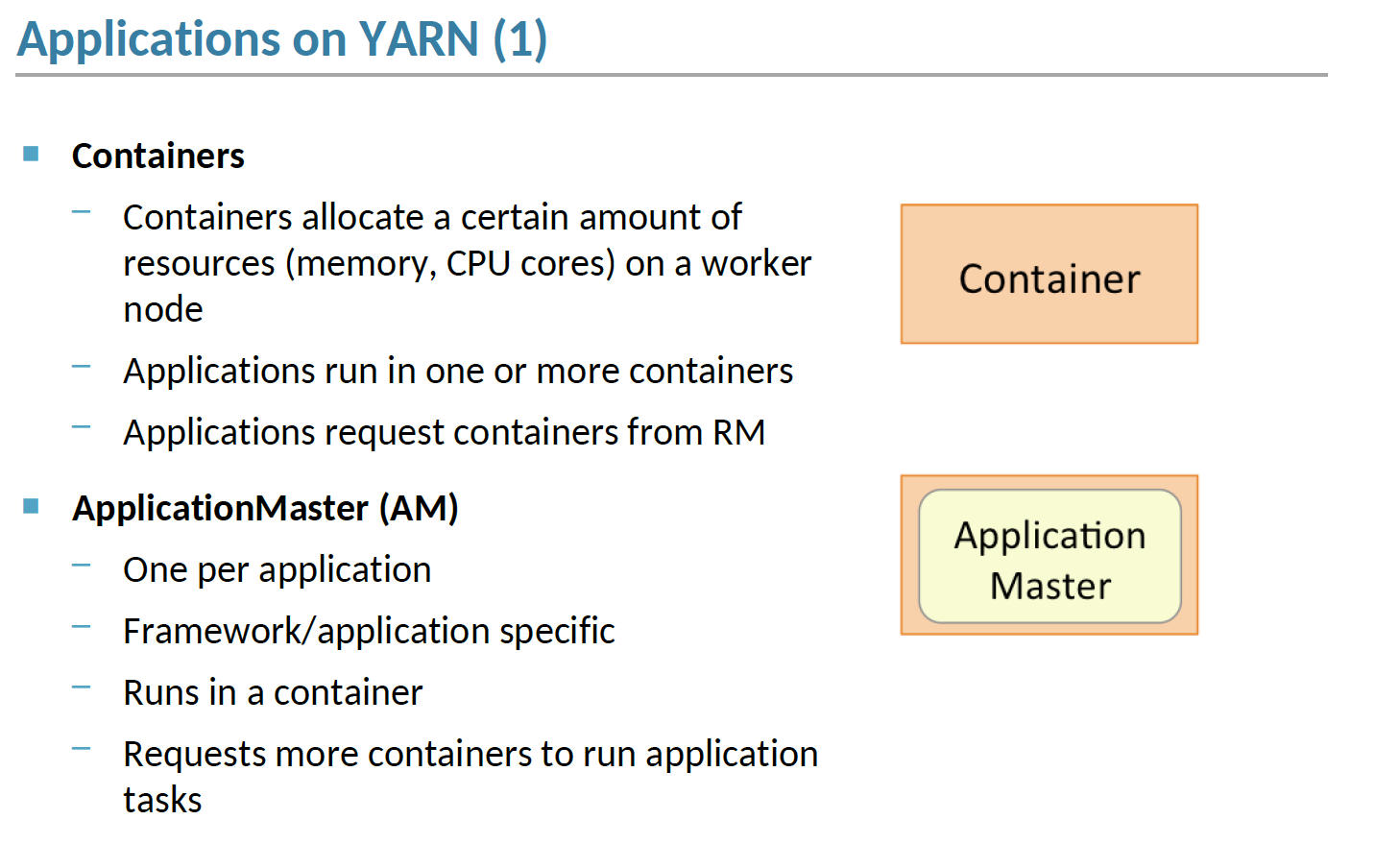
▪ YARN = Yet Another Resource Negotiator

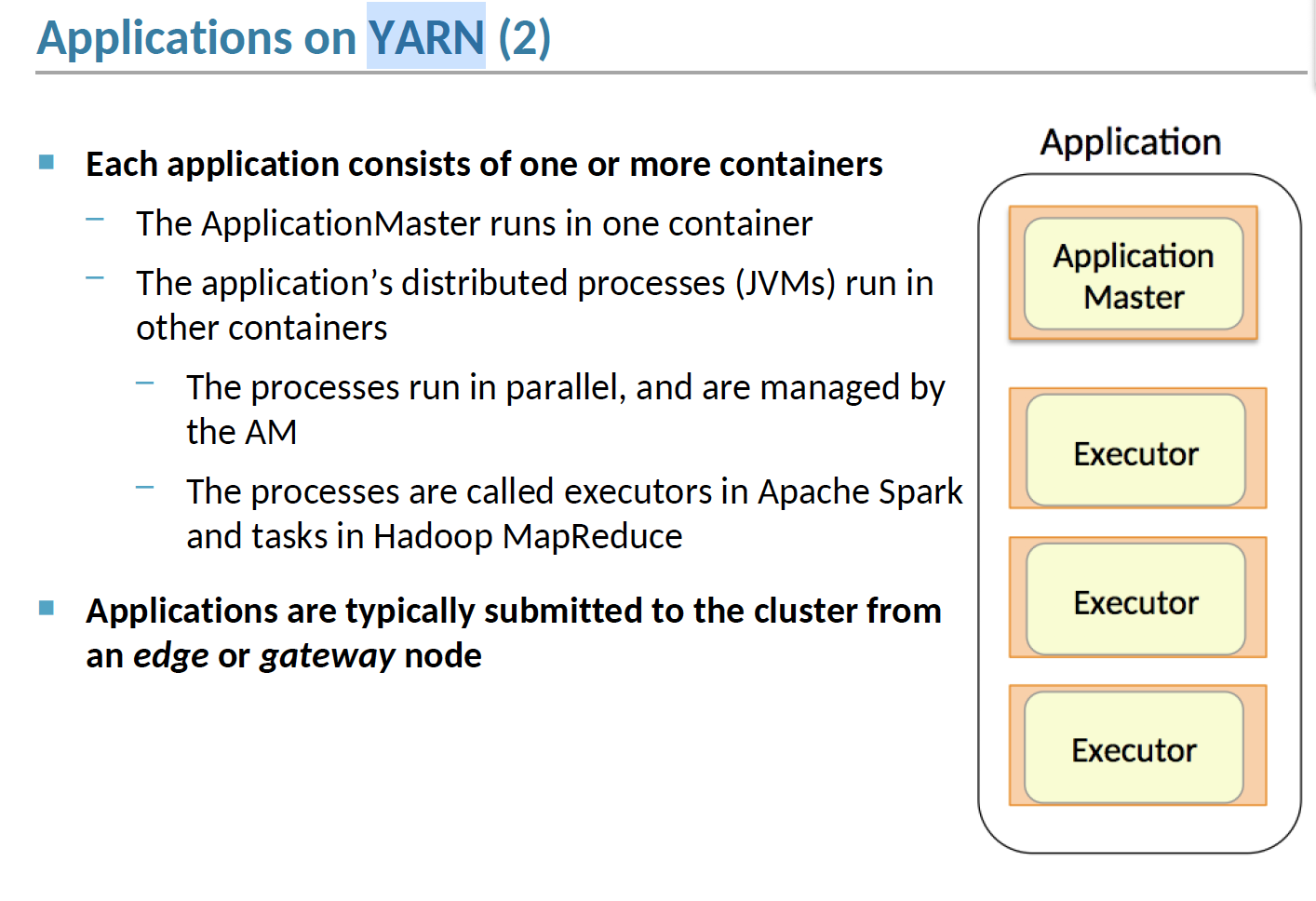
▪ YARN is the Hadoop processing layer that contains

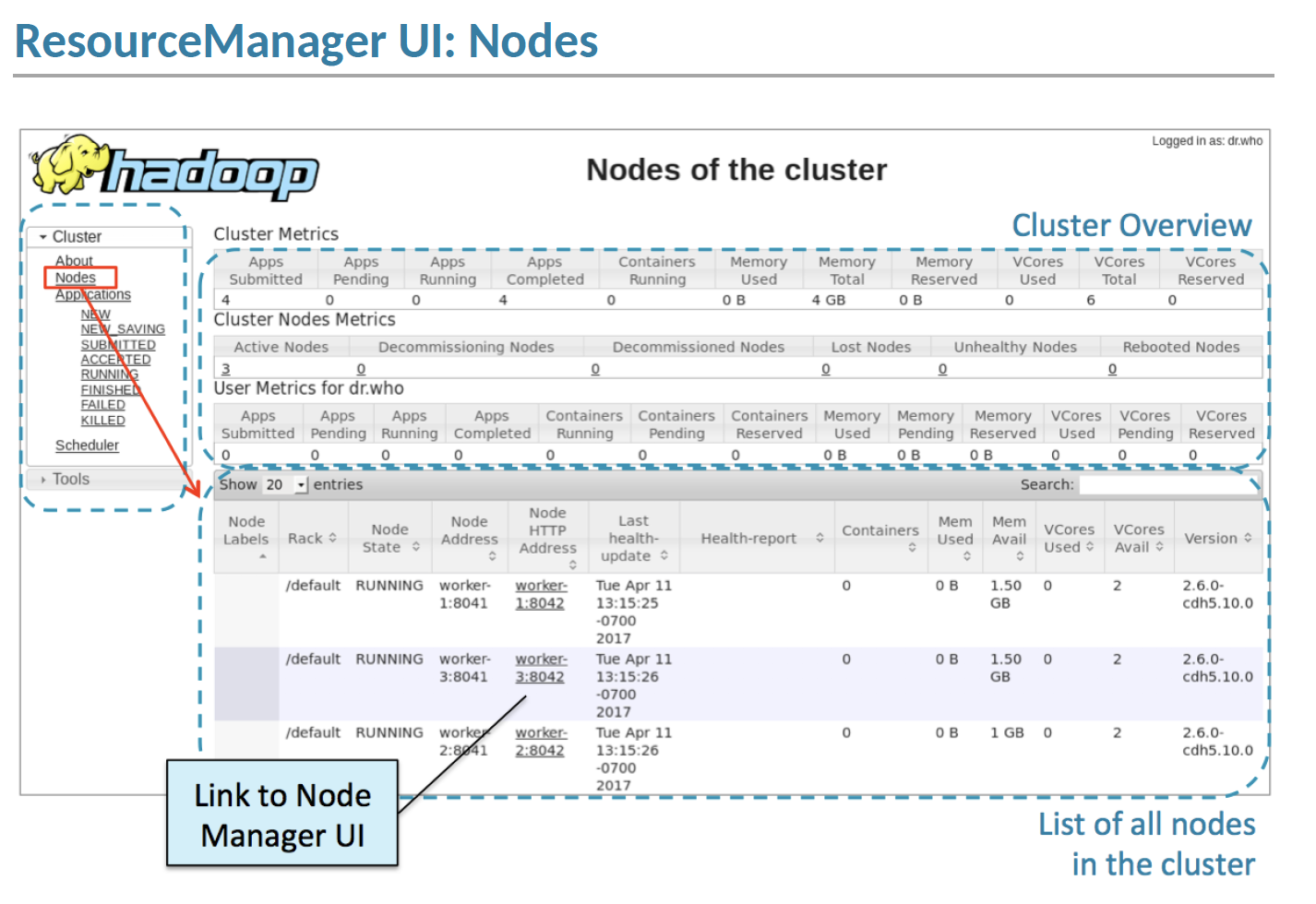
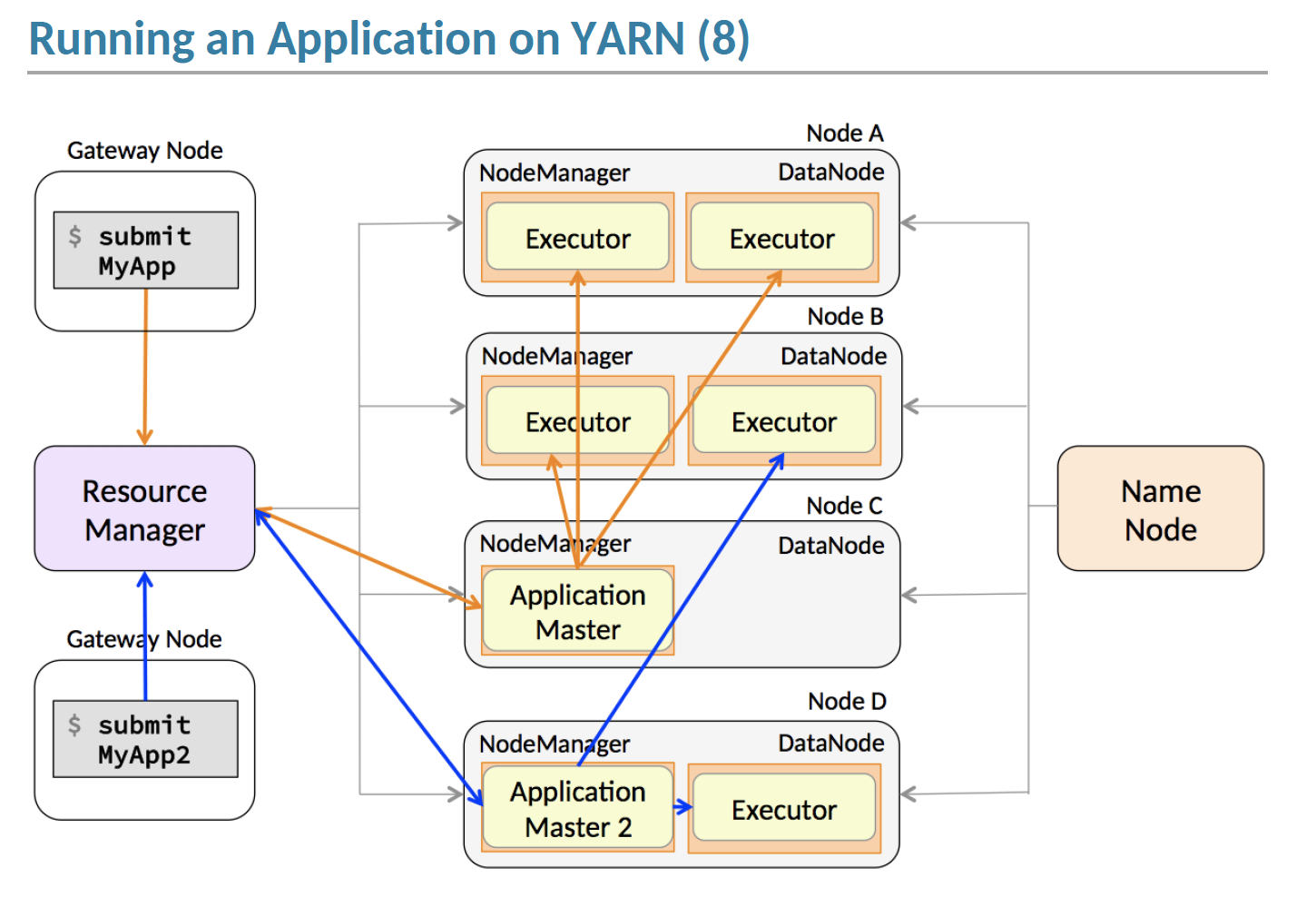
─ A resource manager

─ A job scheduler









<http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html>

**Spark Properties**

**val** conf **=** **new** **SparkConf**()

.setMaster("local[2]")

.setAppName("CountingSheep")

**val** sc **=** **new** **SparkContext**(conf)

Note that we run with local[2], meaning two threads - which represents “minimal” parallelism, which can help detect bugs that only exist when we run in a distributed context.

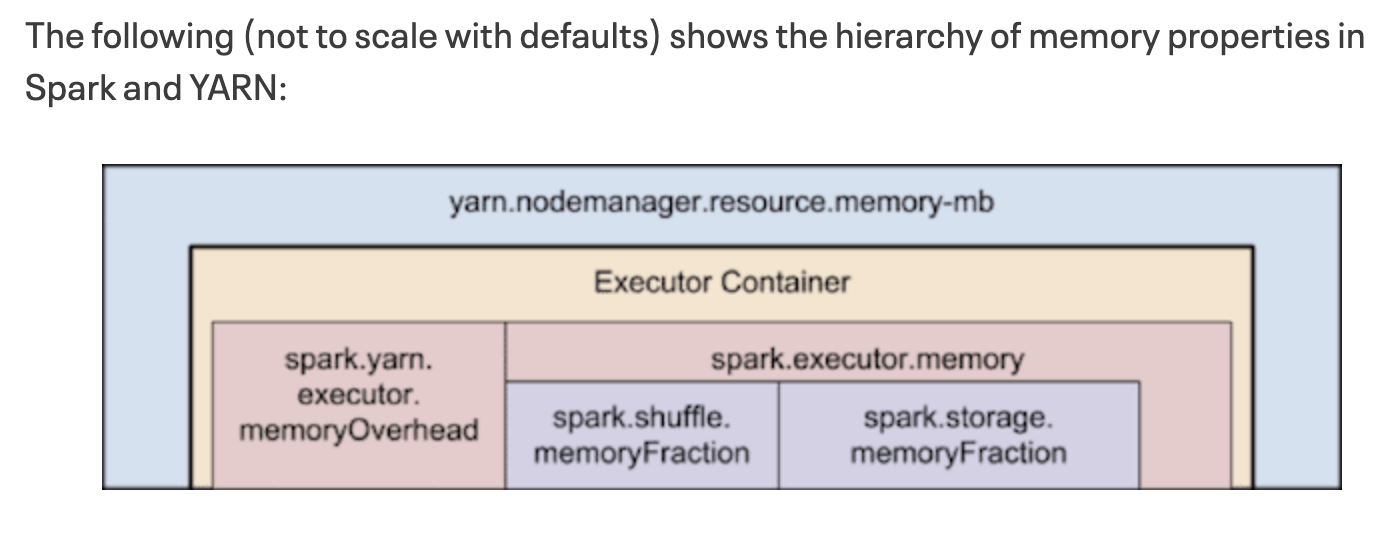
**Application Properties**

|  |  |  |
| --- | --- | --- |
| spark.driver.cores | 1 | Number of cores to use for the driver process, only in cluster mode. |
| spark.driver.memory | 1g | Amount of memory to use for the driver process, i.e. where SparkContext is initialized, in the same format as JVM memory strings with a size unit suffix ("k", "m", "g" or "t") (e.g. 512m, 2g). Note: In client mode, this config must not be set through the SparkConf directly in your application, because the driver JVM has already started at that point. Instead, please set this through the --driver-memory command line option or in your default properties file. |
| spark.driver.memoryOverhead | driverMemory \* 0.10, with minimum of 384 | Amount of non-heap memory to be allocated per driver process in cluster mode, in MiB unless otherwise specified. This is memory that accounts for things like VM overheads, interned strings, other native overheads, etc. This tends to grow with the container size (typically 6-10%). This option is currently supported on YARN, Mesos and Kubernetes. Note: Non-heap memory includes off-heap memory (when spark.memory.offHeap.enabled=true) and memory used by other driver processes (e.g. python process that goes with a PySpark driver) and memory used by other non-driver processes running in the same container. The maximum memory size of container to running driver is determined by the sum of spark.driver.memoryOverhead and spark.driver.memory. |
| spark.executor.memory | 1g | Amount of memory to use per executor process, in the same format as JVM memory strings with a size unit suffix ("k", "m", "g" or "t") (e.g. 512m, 2g). You can make it dynamic with spark.dynamicAllocation.enabled |
| spark.executor.memoryOverhead | executorMemory \* 0.10, with minimum of 384 | Amount of additional memory to be allocated per executor process in cluster mode, in MiB unless otherwise specified. This is memory that accounts for things like VM overheads, interned strings, other native overheads, etc. This tends to grow with the executor size (typically 6-10%). This option is currently supported on YARN and Kubernetes. Note: Additional memory includes PySpark executor memory (when spark.executor.pyspark.memory is not configured) and memory used by other non-executor processes running in the same container. The maximum memory size of container to running executor is determined by the sum of spark.executor.memoryOverhead, spark.executor.memory, spark.memory.offHeap.size and spark.executor.pyspark.memory. |
| spark.master | (none) | The cluster manager to connect to. See the list of [allowed master URL's](https://spark.apache.org/docs/latest/submitting-applications.html#master-urls). |
| spark.executor.cores | 5 | Maximum number of tasks at the same time that an executor can run |

Spark memory required = driver memory + driver memory overhead + (Num. executors)\*(Executor memory+executor memory overhead)

t’s also important to think about how the resources requested by Spark will fit into what YARN has available. The relevant YARN properties are:

* yarn.nodemanager.resource.memory-mb controls the maximum sum of memory used by the containers on each node.
* yarn.nodemanager.resource.cpu-vcores controls the maximum sum of cores used by the containers on each node.



https://blog.cloudera.com/how-to-tune-your-apache-spark-jobs-part-2/