Advanced Spark Configuration

In order to make sure you are using Spark efficiently, there are a few Spark and Datameer properties that advanced users might want to edit.

All properties can added to the **Custom Properties** fields of the cluster or workbook configuration or in the Datameer property files.

- Spark Execution Frameworks
- General Properties
- Spark Port Configuration
- Spark Compression
    - Intermediate compression
    - Output compression
- Auto-Scaling (Dynamic Allocation)
- Spark History Server
- Spark Mode-Specific Configuration
    - SparkSX
    - SparkClient
    - SparkCluster
- Configure Spark Property Overrides
    - Configure thresholds between SparkClient, SparkCluster, and Tez for SparkSX

Spark Execution Frameworks

Set this property to determine which Spark mode you want to use.

Mode	Property	Meaning
SparkClient	`das.execution-framework=SparkClient`	Runs Spark in YARN-client mode
SparkCluster	`das.execution-framework=SparkCluster`	Runs Spark in YARN-cluster mode
SparkSX	`das.execution-framework=SparkSX`	Runs Spark Smart Execution

Each of these works with Datameer Spark auto-scaling and without dynamic resource allocation.

General Properties

The following sub-sections describe properties that apply to all of the Spark execution frameworks. The sections that follow describe properties specific to SparkSX, SparkClient, and SparkCluster frameworks.

Note that the `spark.executor.instances`, `spark.executor.cores` and `spark.executor.memory` properties are performance-related and should be adapted according to the tuning guide.

As of Datameer 6.3

You can autoconfigure the amount of memory used by Spark by setting the `spark.executor.memory` property to auto. Changing this setting means that Datameer calculates the amount of memory to allot for Spark based on the available YARN memory ad vcores and the available Spark executor cores. This new setting helps make sure Spark is getting enough memory and reduces your need to tune Spark.

Property name	Default	Meaning
`spark.executor.instances`	1	The number of executors if dynamic resource allocation is disabled For dynamic mode, refer to auto-scaling
`spark.executor.cores`	1	The number of cores allocated to each executor

`spark.executor.memory`	1g, auto in v6.3	Amount of memory allocated per executor process (e.g. 512m, 2g, 8g) Heap size settings can be set with `spark.executor.memory` Auto allocates the correct amount of memory based on your settings.
`spark.yarn.executor.memoryOverhead`	executorMemory * 0.10, with minimum of 384	The amount of off heap memory (in megabytes) allocated per executor. This memory that accounts for things such as VM overheads, interned strings, and other native overheads. This tends to grow with the executor size (typically 6-10%).

Don't change the following properties unless you encounter errors.

Property name	Default	Meaning
`spark.serializer`	org.apache.spark.serializer.KryoSerializer	Class to use for serializing objects that a network or need to be cached in serializ default of Java serialization works with a Java object but is quite slow, so Datame using `org.apache.spark.serializ lizer` and configuring Kryo serializatio necessary. This can be any subclass of `park.Serializer` .
`spark.executor.extraJavaOptions`		A string of extra JVM options to pass to as GC settings or other logging. It is illeg properties or heap size settings using th properties should be set using a SparkC `spark-defaults.conf` file used with script. Heap size settings can be set wit `tor.memory`.
`spark.driver.extraJavaOptions`		A string of extra JVM options to pass to as GC settings or other logging. Note*:* In client mode, this config must n the `SparkConf` directly in your applica the driver JVM has already started at th set this through the `--driver-java-` and line option or in your default propert
`spark.yarn.am.extraJavaOptions`		A string of extra JVM options to pass to Application Master in client mode. In clu `spark.driver.extraJavaOptions`
`spark.driver.extraClassPath`		Extra classpath entries to prepend to th the driver. Note*:* In client mode, this configuration through the `SparkConf` directly in you because the driver JVM has already sta Instead, set this through the `--driver` command line option or in your default p Can be used to specify the classpath fo libraries.
`spark.executor.extraClassPath`		Extra classpath entries to prepend to th executors. This ensures backwards-con older versions of Spark. Users typically to set this option. Can be used to specify the classpath fo

`spark.driver.extraClassPath`		Extra classpath entries to prepend to th the driver. Note: In client mode, this configuration through the `SparkConf` directly in you because the driver JVM has already sta Instead, set this through the `--driver` command line option or in your default p
`spark.yarn.am.extraLibraryPath`		Set a special library path to use when la YARN Application Master in client mode this propertycan be used to set a path to native libraries.
`spark.executor.extraLibraryPath`		Set a special library path to use when la executor JVMs. For example, this prope to set a path tothe cluster's native librari
`spark.driver.extraLibraryPath`		Set a special library path to use when la driver JVM. For example, this propertyc a path to the cluster's native libraries. Note : In client mode, this config must r through the `SparkConf` directly in you because the driver JVM has already sta Instead, set this through the `--driver` h command line option or in your defau
`das.spark.launcher.spark-submit-opts`	-XX:MaxPermSize=48m -Xmx16m	JVM options for the spark-submit proces the SparkCluster framework. Enables overriding JVM options for the process launched in the SparkCluster fr
`das.spark.launcher.app.status.poll.interval`	10s	Spark Launcher polls application status, between app status fetching can be set property.
`das.spark.context.max-idle-time`	100ms	Max idle time to keep the SparkClient a running. The Spark context times out after this a remaining idle.
`das.splitting.map-wave-count`	6	The count to influence the number of ma uses for internal and for file-based impo Determines how many times a single Ha use the available map slots.
`das.map-tasks-per-node`	10	Used as a foundation for calculating the of splits based on the number of physica

Spark Port Configuration

> Don't change these properties unless you encounter errors.

Property name	Default	Meaning
`spark.ui.port`	4040	Port for your application's dashboard, which shows memory and workload data
`spark.driver.port`	0	Port for the driver to listen on, which is used for communicating with the executors and the standalone master
`spark.replClassServer.port`	0	Port for the driver's HTTP class server to listen on, which is only relevant for the Spark shell
`spark.executor.port`	0	Port for the executor to listen on, which is used for communicating with the driver
`spark.fileserver.port`	0	Port for the driver's HTTP file server to listen on

`spark.broadcast.port`	0	Port for the driver's HTTP broadcast server to listen on, which is not relevant for torrent broadcast
`spark.blockManager.port`	0	Port for all block managers to listen on, which exists on both the driver and the executors
`spark.yarn.am.port`	0	Port for the YARN application ,aster to listen on. In YARN client mode, this is used to communicate between the Spark driver running on a gateway and the YARN application master running on YARN. In YARN cluster mode, this is used for the dynamic executor feature, where it handles the kill from the scheduler backend.
`spark.port.maxRetries`	16	Maximum number of retries when binding to a port before giving up. When a port is given a specific value (non-0), each subsequent retry increments the port used in the previous attempt by 1 before retrying. This allows it to try a range of ports from the start port specified to port + maxRetries.

1. If the port is chosen randomly (value 0), Spark takes a port between 1024 and 65535 (inclusive) and also tries a range of ports from the start port specified to port + maxRetries.
2. SparkCluster framework opens a server port on the Datameer conductor instance for launched applications to connect back and report status. The launcher server listens on the localhost only. This port is hard-coded to zero and can't be overridden. This is passed to the launched app using the environment variable "_SPARK_LAUNCHER_PORT", in case you need to inspect this to help debug any connection issues.

Spark Compression

Intermediate compression

Changing the `compression.codec` to Snappy could lead to performance improvements, but you need to make sure the Snappy libraries are available on each cluster node.

Property Name	Default	Meaning
`spark.io.compression.codec`	lzf	Used to compress internal data such as RDD partitions, broadcast variables, and shuffle outputs. By default, Spark provides three codecs: `lz4`, `lzf`, and `snappy`. You can also use fully qualified class names to specify the codec, e.g. `org.apache.spark.io.LZ4CompressionCodec`, `org.apache.spark.io.LZFCompressionCodec`, and `org.apache.spark.io.SnappyCompressionCodec`.
`spark.io.compression.lz4.blockSize`	32k	Block size used in LZ4 compression. Lowering this block siz also lowers shuffle memory usage when LZ4 is used.
`spark.io.compression.snappy.blockSize`	32k	Block size used in Snappy compression. Lowering this block size also lowers shuffle memory usage when Snappy is used.
`spark.rdd.compress`	false	Determines whether to compress serialized RDD partitions (e.g. for `StorageLevel.MEMORY_ONLY_SER`). Can save substantial space at the cost of extra CPU time.
`spark.broadcast.compress`	true	Determines whether to compress broadcast variables before sending them, which is recommended.
`spark.shuffle.compress`	true	Determines whether to compress map output files, which is recommended. Compression uses `spark.io.compression.codec`.
`spark.shuffle.spill.compress`	true	Determines whether to compress data spilled during shuffles. Compression uses `spark.io.compression.codec`.

Output compression

Example for Apache distributions

```
hadoop.mapred.output.compress=true
hadoop.mapred.output.compression.codec=org.apache.hadoop.io.compress.Defau
ltCodec
```

or

Example for other distributions

```
hadoop.mapred.output.compress=true
hadoop.mapred.output.compression.codec=org.apache.hadoop.io.compress.Snapp
yCodec
```

Auto-Scaling (Dynamic Allocation)

To remove the need for the cluster-wide NodeManager installations usually required by Spark, Datameer implemented our own dynamic resource allocation logic for Spark. If you want, you can still run Spark using their native implementation if you install the NodeManager plug-in and configure it for Datameer using custom properties. Because Datameer uses its own implementation, SparkClient only deallocates executors when there aren't Datameer jobs running within SparkContext, and SparkCluster only deallocates executors between Spark jobs if the Datameer job includes multiple Spark jobs. Datameer's dynamic allocation is enabled by default.

These properties should be aligned with your cluster settings.

Property Name	Default	Meaning
`das.spark.context.auto-scale-enabled`	true	Whether Datameer should dynamically scale up or down Spark containers. This property should typically be turned on in order to free up unneeded cluster resources. When enabled, the Spark context auto-scales executors based on load.
`datameer.yarn.available-node-vcores` `das.yarn.available-node-vcores` in v6.3	8, auto in v6.3	The amount of cluster cores allocated for Spark auto-scaling.
`datameer.yarn.available-node-memory` `das.yarn.available-node-memory` in v6.3	8g, auto in v6.3	The amount of cluster memory allocated for Spark auto-scaling. Auto-scaling also makes sure that the overhead is included into the calculation.
`spark.executor.instances`	4	Minimum number of Spark executors to kept up per Datameer SparkClient application. These are also kept up when all executors are idle. One Spark executor is equivalent to one YARN container.
`das.spark.context.max-executors`	-1	Maximum number of Spark executors to use per Spark job. One Spark executor is equivalent to one YARN container. The Spark context allocates executors up to this amount when required. < 0 means no max

As of Datameer 6.3

The following properties were renamed and now are set to auto:

- `datameer.yarn.available-node-memory` to `das.yarn.available-node-memory`
- `datameer.yarn.available-node-vcores` to `das.yarn.available-node-vcores`

Setting these to auto uses the available memory or vcores for the node on the cluster with the lowest memory or vcore configuration. This new setting option means you no longer have to configure the properties manually based on your Datameer configuration. When the properties are set to auto, they logs cluster values, while if they are set to a specific value, they log the configured amount.

Spark History Server

Property name	Default	Meaning
`spark.eventLog.enabled`	false	Whether to log Spark events. This property is useful for reconstructing the web UI after the application has finished.
`spark.eventLog.dir`	file:///tmp/spark-events	Base directory in which Spark events are logged, if `spark.eventLog.enabled` is set to true. Within this base directory, Spark creates a sub-directory for each application and logs the events specific to the application in this directory. Users might want to set this to a unified location like an HDFS directory so history files can be read by the history server.

Further information can be found under Troubleshooting.

Spark Mode-Specific Configuration

SparkSX

For default and suggested behavior, changes are not recommended.

Property name	Default	Meaning
`das.sparksx.small.max-uncompressed-size`	10g	Max data input size threshold for small data sets in SparkSX
`das.sparksx.small.execution-framework`	SparkClient	SparkSX framework for small data sets
`das.sparksx.medium.max-uncompressed-size`	100g	Max data input size threshold for medium data sets in SparkSX
`das.sparksx.medium.execution-framework`	SparkCluster	SparkSX framework for medium data sets
`das.sparksx.large.execution-framework`	Tez	SparkSX framework for large data sets
`das.sparksx.unknown.execution-framework`	SparkCluster	SparkSX default framework for cases when data input is an unknown size

SparkClient

In SparkClient mode, the driver runs inside the Datameer JVM and the application master is only used for requesting resources from YARN. B e cause the driver has already started at that point, the driver properties must not be set and application master properties should be set instead.

Don't change these properties unless you encounter errors.

Property name	Spark default	Meaning
spark.yarn.am.cores	1	Number of cores allocated for the YARN Application Master
spark.yarn.am.memory	2g	Amount of memory allocated for the YARN Application Master
spark.yarn.am.memoryOverhead	AM memory * 0.10, with minimum of 384	The amount of off heap memory (in megabytes) allocated for YARN Application Master

SparkCluster

In cluster mode, the Spark driver runs inside an application master process which is managed by YARN on the cluster. You can specify the size of the AM in cluster mode using the driver properties.

> Don't change these properties unless you encounter errors.

Property name	Spark default	Meaning
spark.driver.cores	1	Number of cores allocated for the driver process
spark.driver.memory	2g	Amount of memory allocated for the driver process
spark.yarn.driver.memoryOverhead	driverMemory * 0.10, with minimum of 384	The amount of off heap memory (in megabytes) allocated per driver

Configure Spark Property Overrides

You can set different scaling settings for SparkClient and SparkCluster by overriding them on a per-execution basis, as in the following example:

```
spark.executor.cores=1
framework.sparkclient.das.spark.context.auto-scale-enabled=false
framework.sparkclient.spark.executor.instances=5
framework.sparkclient.spark.executor.memory=16g
framework.sparkcluster.das.spark.context.auto-scale-enabled=true
framework.sparkcluster.spark.executor.instances=1
framework.sparkcluster.spark.executor.memory=20g
```

The above uses a fixed number of executors for the smaller workloads and uses dynamic scaling for larger workloads. The `spark.executor.cores` property is shared across both frameworks.

This ability is currently only supported with the SparkClient and SparkCluster execution frameworks and doesn't work with other execution frameworks, such as Tez or MapReduce.

Configure thresholds between SparkClient, SparkCluster, and Tez for SparkSX

Change the following properties in Tez to configure when Tez switches to Spark:

```
SparkClient max
das.execution-framework.sparkclient.max-uncompressed.bytes.mb=10240
SparkCluster max
das.execution-framework.sparkcluster.max-uncompressed.bytes.mb=102400
```