`spark-submit`

`spark-submit` is the command used to submit applications to a Spark cluster. It is a powerful tool that allows you to configure various settings for your Spark jobs, including memory and CPU allocation, cluster modes, and application-specific parameters. Properly configuring `spark-submit` is essential for optimizing Spark jobs for performance and resource usage.

Key Configurations in `spark-submit`

Below are the most important configurations you can use with `spark-submit`, along with their purposes and examples:

1. Application Resource Configuration

- **`--master`**: Specifies the cluster manager to connect to. It can be `local` for local mode, `yarn` for Hadoop YARN, `mesos` for Apache Mesos, or `k8s` for Kubernetes.
  - **Example**: `--master yarn`
- **`--deploy-mode`**: Defines whether to launch the driver on the worker nodes (`cluster`) or locally on the machine submitting the application (`client`).
  - **Example**: `--deploy-mode cluster`
- **`--num-executors`**: Sets the number of executors to use for the job. This is applicable in cluster modes like YARN.
  - **Example**: `--num-executors 5`
- **`--executor-cores`**: Specifies the number of CPU cores per executor. Higher values increase parallelism.
  - **Example**: `--executor-cores 4`
- **`--executor-memory`**: Allocates memory for each executor process. Proper sizing can prevent out-of-memory errors.
  - **Example**: `--executor-memory 8G`
- **`--driver-memory`**: Sets the amount of memory allocated for the driver process.
  - **Example**: `--driver-memory 4G`

2. Configuration for Dynamic Resource Allocation

- **`--conf spark.dynamicAllocation.enabled=true`**: Enables dynamic allocation of executors. Spark will scale the number of executors up and down based on workload.
  - **Example**: `--conf spark.dynamicAllocation.enabled=true`
- **`--conf spark.dynamicAllocation.minExecutors`**: Minimum number of executors to be allocated when dynamic allocation is enabled.
  - **Example**: `--conf spark.dynamicAllocation.minExecutors=2`
- **`--conf spark.dynamicAllocation.maxExecutors`**: Maximum number of executors Spark can allocate.
  - **Example**: `--conf spark.dynamicAllocation.maxExecutors=10`

3. Resource Management and Scheduling

- **`--conf spark.yarn.executor.memoryOverhead`**: Extra memory to be allocated per executor for JVM overheads. This is useful for managing memory more effectively.
  - **Example**: `--conf spark.yarn.executor.memoryOverhead=1024`
- **`--conf spark.scheduler.mode`**: Configures the scheduling mode (`FIFO` or `FAIR`). FAIR scheduling allows jobs to share resources more evenly.
  - **Example**: `--conf spark.scheduler.mode=FAIR`
- **`--conf spark.locality.wait`**: Adjusts the amount of time Spark waits to launch tasks on preferred nodes before scheduling elsewhere. Helps in managing locality.
  - **Example**: `--conf spark.locality.wait=3s`

4. Spark Logging and Debugging

- **`--conf spark.eventLog.enabled=true`**: Enables Spark event logging. This helps in monitoring and debugging by storing event information.
  - **Example**: `--conf spark.eventLog.enabled=true`
- **`--conf spark.eventLog.dir`**: Specifies the directory where the event logs should be stored.
  - **Example**: `--conf spark.eventLog.dir=hdfs:///logs/`
- **`--conf spark.executor.logs.rolling.strategy=time`**: Sets the rolling strategy for executor logs. Useful for managing log file sizes and retention.
  - **Example**: `--conf spark.executor.logs.rolling.strategy=time`
- **`--conf spark.executor.logs.rolling.time.interval=daily`**: Defines the interval for rolling executor logs.
  - **Example**: `--conf spark.executor.logs.rolling.time.interval=daily`

5. Application Specific Configurations

- **`--conf spark.sql.shuffle.partitions`**: Configures the number of partitions to use when shuffling data during Spark SQL operations. Tweaking this number can optimize shuffling.
  - **Example**: `--conf spark.sql.shuffle.partitions=200`
- **`--conf spark.serializer`**: Specifies the serializer for RDDs. The default is Java serialization, but Kryo serialization can be more efficient.
  - **Example**: `--conf spark.serializer=org.apache.spark.serializer.KryoSerializer`
- **`--conf spark.executor.extraJavaOptions`**: Passes additional JVM options for executors. Useful for setting system properties or managing JVM memory.
  - **Example**: `--conf spark.executor.extraJavaOptions="-XX:+UseG1GC"`

6. Security Configurations

- **`--conf spark.authenticate=true`**: Enables authentication for Spark communication to enhance security.
  - **Example**: `--conf spark.authenticate=true`
- **`--conf spark.authenticate.secret`**: Defines the secret key for Spark authentication.
  - **Example**: `--conf spark.authenticate.secret=mySecretKey`
- **`--conf spark.ssl.enabled=true`**: Enables SSL for all Spark communication.
  - **Example**: `--conf spark.ssl.enabled=true`

Example `spark-submit` Command

Below is an example of a `spark-submit` command using several of these configurations:

```
spark-submit \
  --master yarn \
  --deploy-mode cluster \
  --num-executors 5 \
  --executor-cores 4 \
  --executor-memory 8G \
  --driver-memory 4G \
  --conf spark.dynamicAllocation.enabled=true \
  --conf spark.dynamicAllocation.minExecutors=2 \
  --conf spark.dynamicAllocation.maxExecutors=10 \
  --conf spark.yarn.executor.memoryOverhead=1024 \
  --conf spark.scheduler.mode=FAIR \
  --conf spark.eventLog.enabled=true \
  --conf spark.eventLog.dir=hdfs:///logs/ \
  --conf spark.sql.shuffle.partitions=200 \
  --conf spark.serializer=org.apache.spark.serializer.KryoSerializer \
  --conf spark.executor.extraJavaOptions="-XX:+UseG1GC" \
  --conf spark.authenticate=true \
  --conf spark.authenticate.secret=mySecretKey \
```

```
--class com.example.MySparkApp \
/path/to/my-spark-app.jar
```