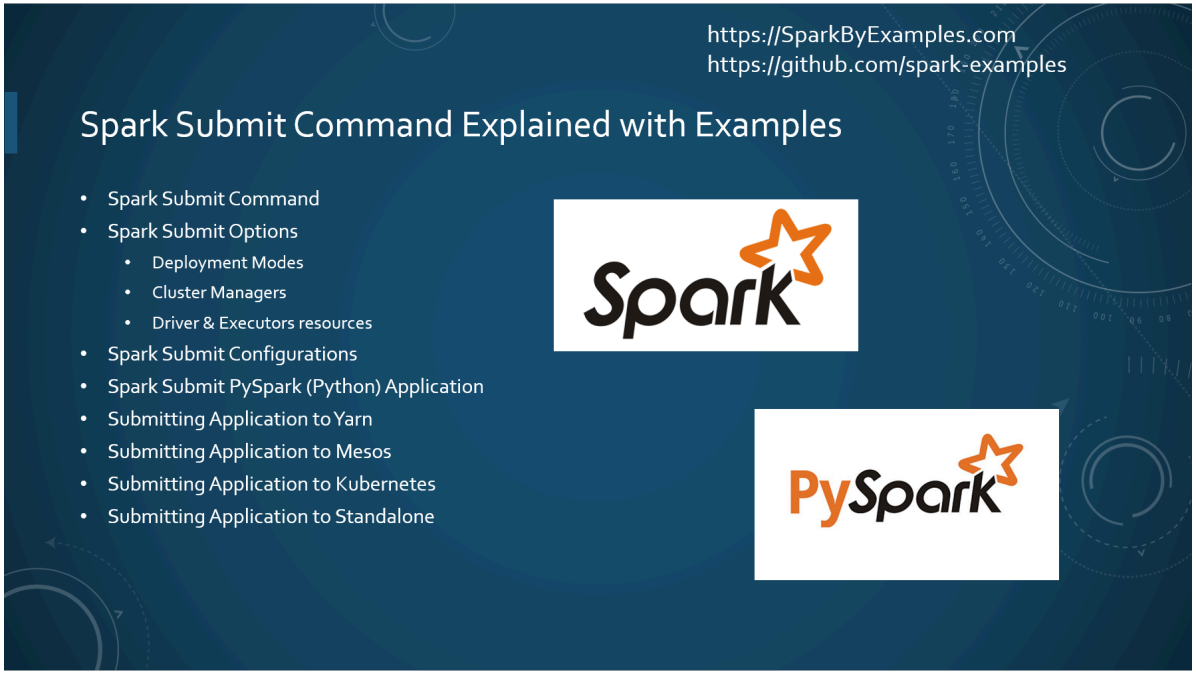
20 mins read



The spark-submit command is a utility for executing or submitting Spark, PySpark, and SparklyR jobs either locally or to a cluster. In this comprehensive guide, I will explain the spark-submit syntax, different command options, advanced configurations, and how to use an uber jar or zip file for Scala and Java, use Python .py file, and finally, submit the application on Yarn, Mesos, Kubernetes, and standalone cluster managers. I will cover everything you need to know to run your applications successfully using the spark-submit command.

At a high level, you need to know that the spark-submit command supports the following cluster managers and deployment modes.

- **Cluster Managers:** Standalone, Yarn, Kubernetes, and Mesos.
- **Deployment Modes:** `client` or `cluster`

Related:

- How to Submit Spark Application via REST API

- How to Debug Spark Application Running Locally or Remote

Table of contents

1. Spark Submit Command

Spark binary comes with `spark-submit.sh` script file for Linux, Mac, and `spark-submit.cmd` command file for Windows, these scripts are available at `$SPARK_HOME/bin` directory.

If you are using Cloudera distribution, you may also find `spark2-submit.sh` which is used to run Spark 2.x applications. By adding this Cloudera supports both Spark 1.x and Spark 2.x applications to run in parallel.

spark-submit command internally uses `org.apache.spark.deploy.SparkSubmit` class with the options and command line arguments you specify.

Below is a spark-submit command with the most-used command options.

```
./bin/spark-submit \
  --master ‹master-url› \
  --deploy-mode ‹deploy-mode› \
  --conf ‹key‹=‹value› \
  --driver-memory ‹value›g \
  --executor-memory ‹value›g \
  --executor-cores ‹number of cores›  \
  --jars  ‹comma separated dependencies›
  --class ‹main-class› \
  ‹application-jar› \
```

```
  [application-arguments]
```

You can also submit the application like below without using the script.

```
./bin/spark-class org.apache.spark.deploy.SparkSubmit <options & arguments>
```

2. Spark Submit Options

Spark-submit options are parameters that you can specify when submitting a Spark application using the spark-submit command. These options are essential for configuring the Spark application according to its requirements, such as resource allocation, dependency management, and execution mode. They provide flexibility and control over how the application is run and how resources are utilized within the Spark cluster.

```
# Spark submit help
```

```
cd $SPARK_HOME
./bin/spark-submit --help
```

2. 1 Deployment Modes (–deploy-mode)

Use `--deploy-mode` option to specify whether to run the spark application in `client` mode or `cluster` mode. In client mode Spark runs the driver program from the current system and in cluster mode, it runs the driver program in the cluster.

VALUE	DESCRIPTION
cluster	In cluster mode, the driver program runs independently of the client that submitted the job. Once the driver program is launched on a cluster node, the client disconnects, and the driver communicates with the cluster manager to request resources and coordinate the execution of tasks. This mode is typically used for production deployments
client	In client mode, the client machine maintains an active connection to the cluster manager throughout the execution of the Spark application. This allows the client to monitor the progress of the application, view logs, and receive status updates in real-time. Note that in client mode only the driver runs locally and all other executors run on the cluster.

2.2 Cluster Managers (–master)

Using `--master` option, specify the URL of the Spark cluster that the application should connect to. It determines the cluster manager to use and the mode in which the application will be deployed.

CLUSTER MANAGER	VALUE	DESCRIPTION
Yarn	yarn	For running Spark on a Hadoop YARN cluster, you can specify `yarn` as the master. This allows Spark to integrate with YARN for resource management.
Mesos	mesos://HOST:PORT	If you're using Apache Mesos as the cluster manager, you can specify `mesos://HOST:PORT` as the master URL. This enables Spark to interact with Mesos to manage cluster resources.
Standalone	spark://HOST:PORT	If you're running a Spark standalone cluster, you can specify the master URL as `spark://HOST:PORT`, where `HOST:PORT` is the address of the master node of the Spark cluster. This mode is suitable for deploying Spark on a dedicated cluster.
Kubernetes	k8s://HOST:PORT k8s://https://HOST:PORT	Use `k8s://HOST:PORT` for Kubernetes, replace the host and port of Kubernetes.
local	local local[k] local[K,F]	– Use local to run locally, where all computation happens on a single machine with a single

CLUSTER MANAGER	VALUE	DESCRIPTION
		JVM. – Use local[k] where K specified the number of worker threads to use. – Use local[k,F], specify F with number of attempts it should run when fail.

Spark submit cluster managers

Example: Below submits applications to yarn managed cluster.

```
./bin/spark-submit \
    --deploy-mode cluster \
    --master yarn \
    --class org.apache.spark.examples.SparkPi \
    /spark-home/examples/jars/spark-examples_versionxx.jar 80
```

Value 80 on the above example is a command-line argument for the spark program `SparkPi`. The above example calculates a `PI` value of 80.

2.3 Driver and Executor Resources (Cores & Memory)

Use the following options to specify how much memory and cores you want to give for driver and executors.

OPTION	DESCRIPTION
–driver-memory	Specify the driver memory. You can increase this when you collecting large data.
–driver-cores	Number of cores to be used by Spark driver.
–executor-memory	Specify the amount of memory to be used by executor.
–executor-cores	Specify the number of CPU cores to be used by executor.
–num-executors	Number of executors to use by your Spark application.
–total-executor-cores	The total number of executor cores to use.

Example:

```
./bin/spark2-submit \
   --master yarn \
   --deploy-mode cluster \
   --driver-memory 8g \
   --executor-memory 16g \
   --executor-cores 2  \
   --class org.apache.spark.examples.SparkPi \
```

```
    /spark-home/examples/jars/spark-examples_versionxx.jar 80
```

2.4 Other Options

OPTIONS	DESCRIPTION
–files	Specify files by comma separator you want to use in your Spark application. Usually, these can be files from your resource folder.
–verbose	Displays the verbose information. Writes all configurations Spark application uses to the log file.
–config	To specify the memory setting and runtime environment variables and more.

Note: Files specified with `--files` are uploaded to the cluster.

2.5 Spark Submit Example

The following example submits the Spark with Scala/Java application to yarn cluster manager. This application uses 8g driver memory, 16g, and 2 cores for each executor.

```
# Spark submit example
./bin/spark2-submit \
   --verbose
```

```
    --master yarn \
    --deploy-mode cluster \
    --driver-memory 8g \
    --executor-memory 16g \
    --executor-cores 2  \
    --class org.apache.spark.examples.SparkPi \
    /spark-home/examples/jars/spark-examples_versionxx.jar 80
```

3. Spark Submit Configurations

The `--config` option allows you to specify configuration properties for the Spark application. These configuration properties can be used to customize various aspects of the Spark runtime environment, such as memory settings, parallelism, logging, and more.

CONFIGURATION KEY	**CONFIGURATION DESCRIPTION**
spark.sql.shuffle.partitions	Total number of partitions to create for wider shuffle transformations, such as joins and aggregations.
spark.executor.memoryOverhead	The additional memory allocated per executor process in cluster mode, typically reserved for JVM overheads. (Not applicable for PySpark)
spark.serializer	`org.apache.spark.serializer.JavaSerializer` (default) `org.apache.spark.serializer.KryoSerializer`
spark.sql.files.maxPartitionBytes	Specifies the maximum amount of bytes allocated for each partition when reading files. The default is set to 128MB.

CONFIGURATION KEY	**CONFIGURATION DESCRIPTION**
spark.dynamicAllocation.enabled	Specifies whether the number of executors should dynamically scale up or down in response to the workload. The default is set to true.
spark.dynamicAllocation .minExecutors	Specifies the minimum number of executors to utilize when dynamic allocation is enabled.
spark.dynamicAllocation .maxExecutors	Specifies the maximum number of executors to utilize when dynamic allocation is enabled.
spark.executor.extraJavaOptions	Specify JVM options like heap memory e.t.c(see example below)

For the complete list, refer to Spark configurations.

Example :

```
# Spark submit with config
./bin/spark2-submit \
--master yarn \
--deploy-mode cluster \
--conf "spark.sql.shuffle.partitions=20000" \
--conf "spark.executor.memoryOverhead=5244" \
--conf "spark.memory.fraction=0.8" \
--conf "spark.memory.storageFraction=0.2" \
--conf "spark.serializer=org.apache.spark.serializer.KryoSerializer" \
--conf "spark.sql.files.maxPartitionBytes=168435456" \
```

```
--conf "spark.dynamicAllocation.minExecutors=1" \
--conf "spark.dynamicAllocation.maxExecutors=200" \
--conf "spark.dynamicAllocation.enabled=true" \
--conf "spark.executor.extraJavaOptions=-XX:+PrintGCDetails -XX:+PrintGCTimeStamps" \
--files /path/log4j.properties,/path/file2.conf,/path/file3.json \
--class org.apache.spark.examples.SparkPi \
/spark-home/examples/jars/spark-examples_repace-spark-version.jar 80
```

You can also set these configurations in `$SPARK_HOME/conf/spark-defaults.conf`, This applies to every Spark application. And can also set using `SparkConf` programmatically.

```
# Using SparkConf
val config = new SparkConf()
config.set("spark.sql.shuffle.partitions","300")
val spark=SparkSession.builder().config(config)
```

If you specify the configuration in all these options, the first preference goes to SparkConf, then –config and then spark-defaults.conf.

4. Submit Scala or Java Application

Regardless of which language you use, most of the options are the same however, there are few options that are specific to a language, for example, to run a Spark application written in Scala or Java, you need to use the additional following options.

OPTION	DESCRIPTION
–jars	If you have all dependency jar's in a folder, you can pass all these jars using this spark submit –jars option. All your jar files should be comma-separated. for example –jars jar1.jar,jar2.jar, jar3.jar.
–packages	All transitive dependencies will be handled when using this command.
–class	Scala or Java class you wanted to run. This should be a fully qualified name with the package for example `org.apache.spark.examples.SparkPi`.

Note: Files specified with `--jars` and `--packages` are uploaded to the cluster.

Example :

```
# Spark submit Java application
./bin/spark-submit \
--master yarn \
--deploy-mode cluster \
--conf "spark.sql.shuffle.partitions=20000" \
--jars "dependency1.jar,dependency2.jar"
--class com.sparkbyexamples.WordCountExample \
spark-by-examples.jar
```

5. Spark Submit PySpark (Python) Application

When you wanted to spark-submit a PySpark application, you need to specify the .py file you wanted to run and specify the .egg file or .zip file for dependency libraries.

Below are some of the options & configurations specific to PySpark application. besides these you can also use most of the options & configs that are covered above.

PYSPARK SPECIFIC CONFIGURATIONS	DESCRIPTION
–py-files	This is used to specify `.py`, `.zip` or `.egg` files.
–config spark.executor.pyspark.memory	Specify the executor memory for the PySpark application.
–config spark.pyspark.driver.python	Python binary executable to use for PySpark in driver.
–config spark.pyspark.python	Specifies the Python binary executable that should be used for PySpark in both the driver and the executors. This property allows you to specify a custom Python binary executable path if the default Python interpreter needs to be overridden.

Note: Files specified with `--py-files` are uploaded to the cluster before it runs the application. You also upload these files ahead and refer them in your PySpark application.

Example 1 :

```
# Spark submit pyspark application
./bin/spark-submit \
   --master yarn \
   --deploy-mode cluster \
   wordByExample.py
```

Example 2 : Below example uses other python files as dependencies.

```
./bin/spark-submit \
   --master yarn \
   --deploy-mode cluster \
   --py-files file1.py,file2.py,file3.zip
   wordByExample.py
```

6. Submitting Application to Mesos

Here, we are submitting spark application on a Mesos-managed cluster using deployment mode with 5G memory and 8 cores for each executor.

```
# Running Spark application on Mesos cluster manager
./bin/spark-submit \
```

```
  --class org.apache.spark.examples.SparkPi \
  --master mesos://192.168.231.132:7077 \
  --deploy-mode cluster \
  --executor-memory 5G \
  --executor-cores 8 \
   http://examples/jars/spark-examples_versionxx.jar 80
```

7. Submitting Application to Kubernetes

The below example runs Spark application on a Kubernetes managed cluster using cluster deployment mode with 5G memory and 8 cores for each executor.

```
# Running Spark application on Kubernetes cluster
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master k8s://192.168.231.132:443 \
  --deploy-mode cluster \
  --executor-memory 5G \
  --executor-cores 8 \
  /spark-home/examples/jars/spark-examples_versionxx.jar 80
```

8. Submitting Application to Standalone

The below example runs Spark application on a Standalone cluster using cluster deployment mode with 5G memory and 8 cores for each executor.

```
# Running Spark application on standalone cluster
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://192.168.231.132:7077 \
  --deploy-mode cluster \
  --executor-memory 5G \
  --executor-cores 8 \
  /spark-home/examples/jars/spark-examples_versionxx.jar 80
```

Happy Learning !!

Administrative Tasks

In addition to submitting Spark applications, Spark Submit also provides several other functionalities. It can be used to run Spark's built-in examples, test Spark applications, and perform other administrative tasks such as deploying and managing Spark clusters.

Conclusion

In conclusion, Spark Submit is a command-line tool that is an integral part of the Spark ecosystem. It allows users to submit Spark applications to a cluster for execution and provides various functionalities such as running examples, testing applications, and performing administrative tasks. By being able to specify configuration parameters and handling the complexities of the distributed environment, Spark Submit makes it easier for users to develop, test, and deploy their Spark applications.