Spark Application Submission Guidelines

Trainer: Udayakumar M

Data Engineering

Introduction

The spark applications can be submitted to a cluster of any type using the spark-submit script. The application submission guide describes how to do this.

The spark-submit script in Spark's bin directory is used to launch applications on a cluster. It can use all of Spark's supported cluster managers through a uniform interface, so you don't have to configure your application especially for each one.

Bundling Your Application's Dependencies

For Python, you can use the --py-files argument of spark-submit to add .py, .zip or .egg files to be distributed with your application. If you depend on multiple Python files we recommend packaging them into a .zip or .egg

Launching Applications with spark-submit

Once a user application is bundled, it can be launched using the bin/spark-submit script. This script takes care of setting up the classpath with Spark and its dependencies, and can support different cluster managers and deploy modes that Spark supports:

```
./bin/spark-submit \
  --class <main-class> \
  --master <master-url> \
  --deploy-mode <deploy-mode> \
  --conf <key>=<value> \
  ... # other options
  <application-jar> \
  [application-arguments]
```

Some of the commonly used options are:

- --class: The entry point for your application (e.g. org.apache.spark.examples.SparkPi)
- --master: The master URL for the cluster (e.g. spark://23.195.26.187:7077)

- --deploy-mode: Whether to deploy your driver on the worker nodes (cluster) or locally as an external client (client) (default: client) †
- --conf: Arbitrary Spark configuration property in key=value format. For values that contain spaces wrap "key=value" in quotes (as shown). Multiple configurations should be passed as separate arguments. (e.g. --conf <key>=<value> --conf <key2>=<value2>)
- application-jar: Path to a bundled jar including your application and all dependencies. The URL must be globally visible inside of your cluster, for instance, an hdfs:// path or a file:// path that is present on all nodes.
- application-arguments: Arguments passed to the main method of your main class, if any

† A common deployment strategy is to submit your application from a gateway machine that is physically co-located with your worker machines (e.g. Master node in a standalone EC2 cluster). In this setup, client mode is appropriate. In client mode, the driver is launched directly within the spark-submit process which acts as a *client* to the cluster. The input and output of the application is attached to the console. Thus, this mode is especially suitable for applications that involve the REPL (e.g. Spark shell).

Alternatively, if your application is submitted from a machine far from the worker machines (e.g. locally on your laptop), it is common to use cluster mode to minimize network latency between the drivers and the executors. Currently, the standalone mode does not support cluster mode for Python applications.

For Python applications, simply pass a .py file in the place of <application-jar>, and add Python .zip, .egg or .py files to the search path with --py-files.

There are a few options available that are specific to the cluster manager that is being used. For example, with a Spark standalone cluster with cluster deploy mode, you can also specify --supervise to make sure that the driver is automatically restarted if it fails with a non-zero exit code. To enumerate all such options available to spark-submit, run it with --help. Here are a few examples of common options:

```
# Run application locally on 8 cores
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
```

```
  --master local[8] \
  /path/to/examples.py \
  100
# Run on a Spark standalone cluster in client deploy mode
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077 \
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.py\
  1000

# Run on a Spark standalone cluster in cluster deploy mode with supervise
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077 \
  --deploy-mode cluster \
  --supervise \
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.py \
  1000

# Run on a YARN cluster in cluster deploy mode
export HADOOP_CONF_DIR=XXX
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master yarn \
  --deploy-mode cluster \
  --executor-memory 20G \
```

```
  --num-executors 50 \
  /path/to/examples.py \
  1000

# Run a Python application on a Spark standalone cluster
./bin/spark-submit \
  --master spark://207.184.161.138:7077 \
  examples/src/main/python/pi.py \
  1000

# Run on a Mesos cluster in cluster deploy mode with supervise
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master mesos://207.184.161.138:7077 \
  --deploy-mode cluster \
  --supervise \
  --executor-memory 20G \
  --total-executor-cores 100 \
  http://path/to/examples.py \
  1000

# Run on a Kubernetes cluster in cluster deploy mode
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master k8s://xx.yy.zz.ww:443 \
  --deploy-mode cluster \
  --executor-memory 20G \
  --num-executors 50 \
  http://path/to/examples.py \
  1000
```

Master URLs

The master URL passed to Spark can be in one of the following formats:

Master URL	Meaning
local	Run Spark locally with one worker thread (i.e. no parallelism at all).
local[K]	Run Spark locally with K worker threads (ideally, set this to the number of cores on your machine).
local[K,F]	Run Spark locally with K worker threads and F maxFailures
local[*]	Run Spark locally with as many worker threads as logical cores on your machine.
local[*,F]	Run Spark locally with as many worker threads as logical cores on your machine and F maxFailures.
local-cluster[N,C,M]	Local-cluster mode is only for unit tests. It emulates a distributed cluster in a single JVM with N number of workers, C cores per worker and M MiB of memory per worker.
spark://HOST:PORT	Connect to the given Spark standalone cluster master. The port must be whichever one your master is configured to use, which is 7077 by default.
spark://HOST1:PORT1,HOST2:PORT2	Connect to the given Spark standalone cluster with standby masters with Zookeeper. The list must have all the master hosts in the high availability cluster set up with Zookeeper. The port must be whichever each master is configured to use, which is 7077 by default.
mesos://HOST:PORT	Connect to the given Mesos cluster. The port must be whichever one your is configured to use, which is 5050 by default. Or, for a Mesos cluster using ZooKeeper, use mesos://zk://.... To submit with --deploy-mode cluster, the HOST:PORT should be configured to connect to the MesosClusterDispatcher.
yarn	Connect to a YARN cluster in client or cluster mode depending on the value of --deploy-mode. The cluster location will be found based on the HADOOP_CONF_DIR or YARN_CONF_DIR variable.
k8s://HOST:PORT	Connect to a Kubernetes cluster in client or cluster mode depending on the value of --deploy-mode. The HOST and PORT refer to the Kubernetes API Server. It connects using TLS by default. In order to force it to use

Master URL	Meaning
	an unsecured connection, you can use k8s://http://HOST:PORT.

Loading Configuration from a File

The spark-submit script can load default Spark configuration values from a properties file and pass them on to your application. By default, it will read options from conf/spark-defaults.conf in the Spark directory. For more detail, see the section on loading default configurations.

Loading default Spark configurations this way can obviate the need for certain flags to spark-submit. For instance, if the spark.master property is set, you can safely omit the --master flag from spark-submit. In general, configuration values explicitly set on a SparkConf take the highest precedence, then flags passed to spark-submit, then values in the defaults file.

If you are ever unclear where configuration options are coming from, you can print out fine-grained debugging information by running spark-submit with the --verbose option.

Advanced Dependency Management

When using spark-submit, the application jar along with any jars included with the --jars option will be automatically transferred to the cluster. URLs supplied after --jars must be separated by commas. That list is included in the driver and executor classpaths. Directory expansion does not work with --jars.

Spark uses the following URL scheme to allow different strategies for disseminating jars:

- **file:** - Absolute paths and file:/ URIs are served by the driver's HTTP file server, and every executor pulls the file from the driver HTTP server.
- **hdfs:**, **http:**, **https:**, **ftp:** - these pull down files and JARs from the URI as expected
- **local:** - a URI starting with local:/ is expected to exist as a local file on each worker node. This means that no network IO will be incurred, and works well for large files/JARs that are pushed to each worker, or shared via NFS, GlusterFS, etc.

Note that JARs and files are copied to the working directory for each SparkContext on the executor nodes. This can use up a significant amount of space over time and will need to be cleaned up. With YARN, cleanup is handled automatically, and with Spark standalone, automatic cleanup can be configured with the spark.worker.cleanup.appDataTtl property.

Users may also include any other dependencies by supplying a comma-delimited list of Maven coordinates with --packages. All transitive dependencies will be handled when using this command. Additional repositories (or resolvers in SBT) can be added in a comma-delimited fashion with the flag --repositories. (Note that credentials for password-protected repositories can be supplied in some cases in the repository URI, such as in https://user:password@host/.... Be careful when supplying credentials this way.) These commands can be used with pyspark, spark-shell, and spark-submit to include Spark Packages.

For Python, the equivalent --py-files option can be used to distribute .egg, .zip and .py libraries to executors.