Spark 2018

Spark SQL

IE BUSINESS SCHOOL - MASTER IN BUSINESS ANALYTICS & BIG DATA

Options when “deploying” an app

- Command line REPLs
- Notebooks
- Rest API
- Cloud SDKs
- spark-submit

Which one to use?

It depends on the use case

- Command line REPLs
- Notebooks

- Rest API
- Cloud SDKs
- spark-submit

Data exploration

Pipeline contruction

Interactive analysis

ML

ETL

Streaming

What happens when you run a Spark application?

- User submits an app either using spark-submit, rest-api or cloudSDK commands. Let's assume we use spark-submit here.
- spark -submit is executed launching the driver program and calling the main() method. Configuration options can be passed in the command line or in the code.If no configuration options are passed, the driver will use the default ones in conf/spark-defaults.conf in the Spark directory.
- The driver contacts the cluster manager to ask for resources
- The cluster manager instantiates executors in the different workers
- The driver execute the program and sends the rights orders to the executors depending on the actions and transformations in your code. The orders sent to the executors are the tasks we saw in previous lessons
- Tasks are executed in the executors instanciated by the cluster manager
- When finished executors will be returned to the cluster manager for other apps to use

What happens when you use a REPL?

Similar to what happens when using spark-submit but the REPL does a few things for you:

- Creates the main program
- Creates the spark configuration
- Creates and start spark context

spark-submit

- --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).
- 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

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

Spark-submit examples

```
# Run application locally on 8 cores
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master local[8] \
  /path/to/examples.jar \
  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.jar \
  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.jar \
  1000
```

```
# Run on a YARN cluster
export HADOOP_CONF_DIR=XXX
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master yarn \
  --deploy-mode cluster \  # can be client for client mode
  --executor-memory 20G \
  --num-executors 50 \
  /path/to/examples.jar \
  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.jar \
  1000
```

What to submit in Python, in Java and in scala

- If your code depends on other projects, you will need to package them alongside your application in order to distribute the code to a Spark cluster. To do this, create an assembly jar (or “uber” jar) containing your code and its dependencies. Both sbt and Maven have assembly plugins. When creating assembly jars, list Spark and Hadoop as provided dependencies; these need not be bundled since they are provided by the cluster manager at runtime. Once you have an assembled jar you can call the bin/spark-submit script as shown here while passing your jar.

- 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.

Master URLs

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 (see spark.task.maxFailures for an explanation of this variable)
`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.
`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.

An app example in Python

```python
from __future__ import print_function

import sys
from operator import add

from pyspark.sql import SparkSession


if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: wordcount <file>", file=sys.stderr)
        exit(-1)

    spark = SparkSession\
        .builder\
        .appName("PythonWordCount")\
        .getOrCreate()

    lines = spark.read.text(sys.argv[1]).rdd.map(lambda r: r[0])
    counts = lines.flatMap(lambda x: x.split(' ')) \
                  .map(lambda x: (x, 1)) \
                  .reduceByKey(add)
    output = counts.collect()
    for (word, count) in output:
        print("%s: %i" % (word, count))

    spark.stop()
```

We need to import the required libraries

We need to define a main

We build a spark session

This is the spark code to be executed by executors

Driver terminates the app and executors are returned to cluster manager

Spark configuration

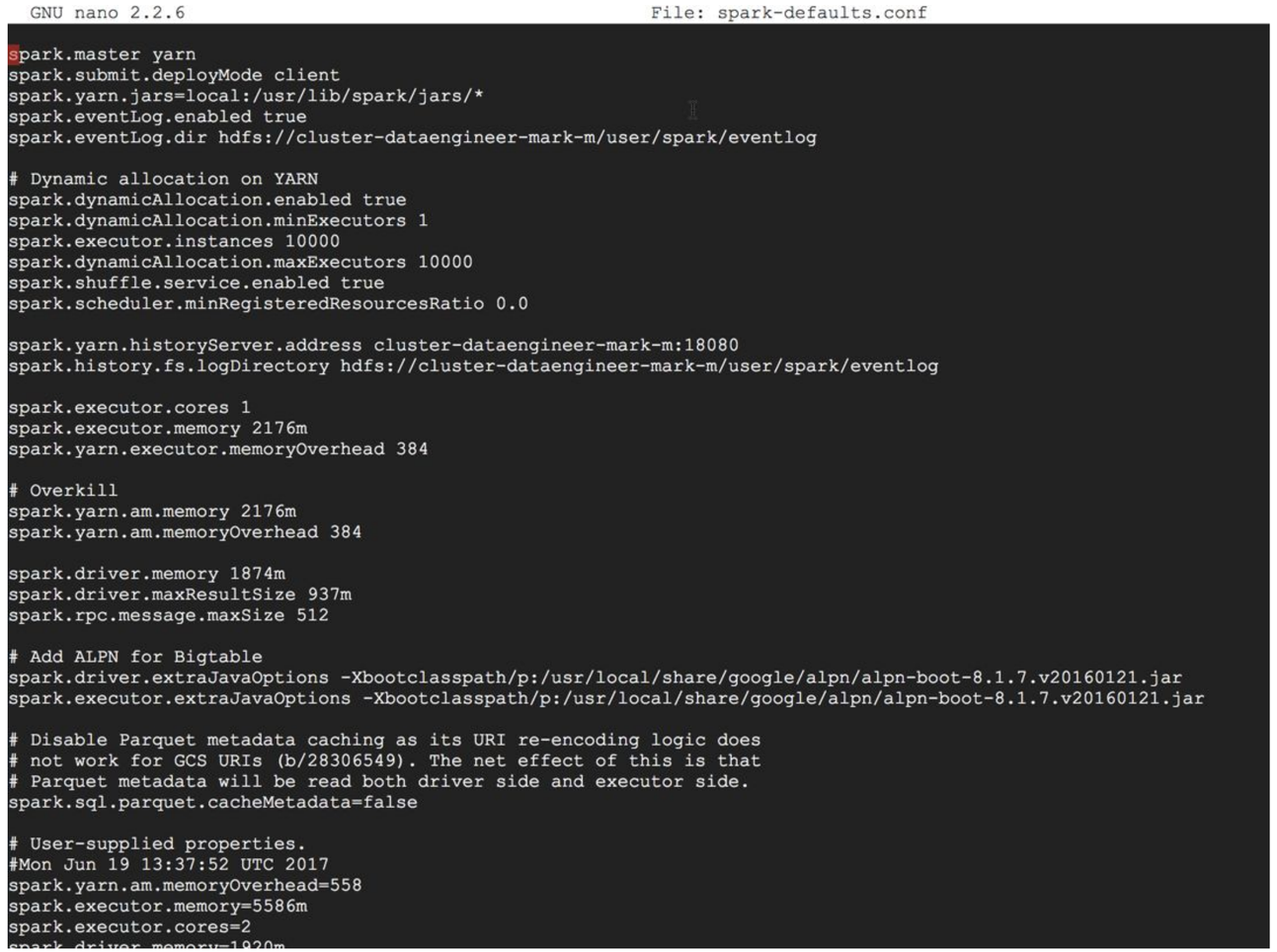
```python
from pyspark import SparkConf, SparkContext


def main():
    conf = SparkConf().set("spark.ui.showConsoleProgress", "false")
    sc = SparkContext(appName="PythonStatusAPIDemo", conf=conf)
```

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

- You can use the defaults (specified in spark configuration files(conf/spark-defaults.conf), pass the configuration via the spark-submit parameters or use SparkConf() in the code.
- An important property to set level of detail when logging: sc.setLogLevel(), for example sc.setLogLevel("Warn"). Valid log levels include: ALL, DEBUG, ERROR, FATAL, INFO, OFF, TRACE, WARN

Spark default configuration



SparkSession

In versions before Spark 2.0:

```
val sparkConf = new SparkConf().setAppName("myapp").setMaster("local"
val sc = new SparkContext(sparkConf).set("spark.some.config.option", "some-value")
```

After 2.0, no need to create the sparkcontext directly, we can do it through a session:

```
val spark = SparkSessio
              .builder(
              .appName("myapp"
              .config("spark.sx.x", myproperty
    .getOrCreate()
spark.conf.set("spark.executor.memory", "2g")
```

In 2.0 spark-shell creates a SparkSession and you can access it via spark

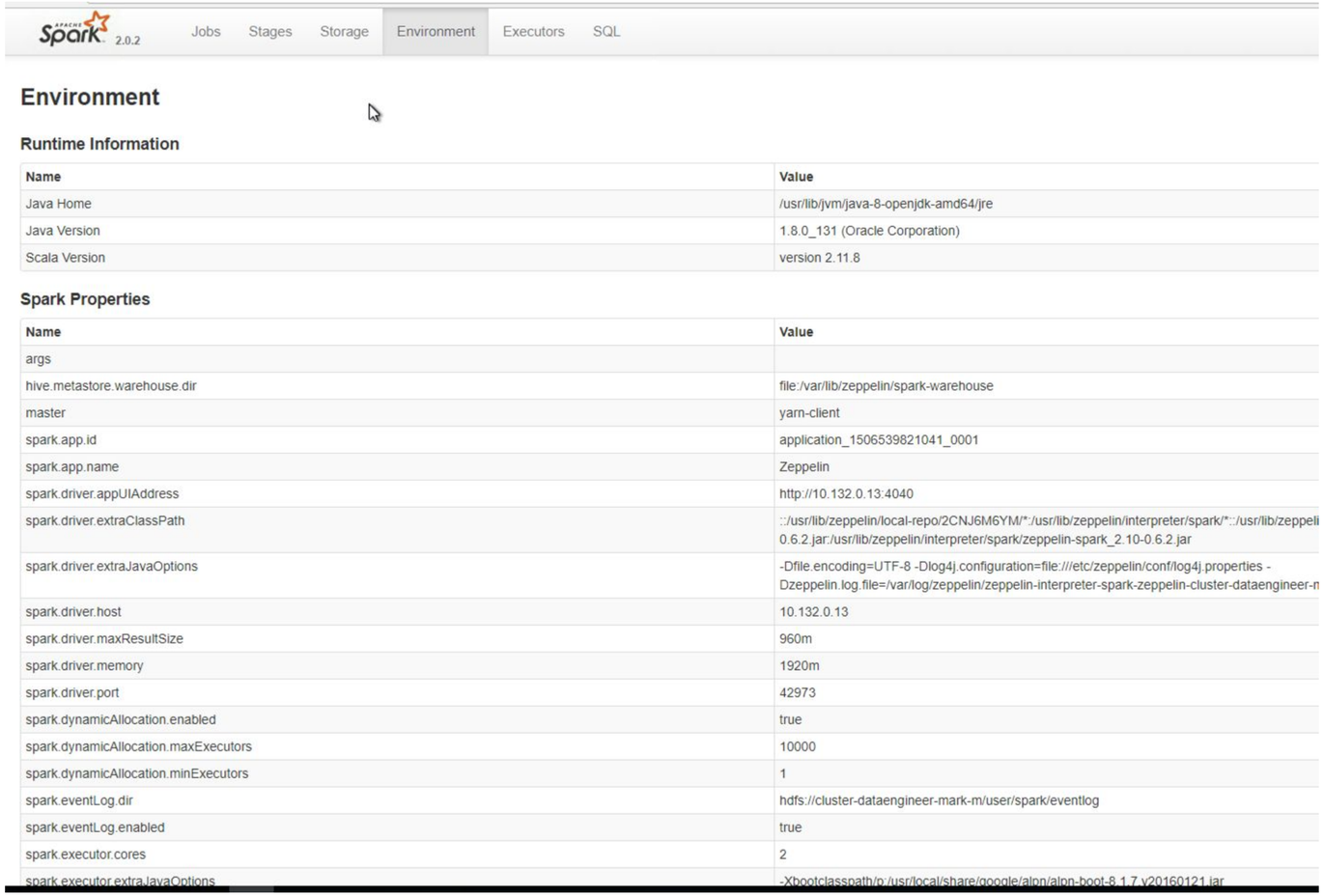
Submitting app options

- Yarn client: driver and spark context are in the computer where we are submitting the program. This is good for development and testing. REPLs tools use this mode
- Yarn cluster: driver and spark context are placed where the application master is. This is good for production apps

Dynamic Allocation

- Executors are not assigned to app the whole time
- Executors not performing anything are returned to cluster
- If many tasks are still in the queue new executors will be requested and assigned to the app
- It requires more scheduling and this could have some impact on performance

Where to get properties for current environment?



Environment

Runtime Information

Name	Value
Java Home	/usr/lib/jvm/java-8-openjdk-amd64/jre
Java Version	1.8.0_131 (Oracle Corporation)
Scala Version	version 2.11.8

Spark Properties

Name	Value
args	
hive.metastore.warehouse.dir	file:/var/lib/zeppelin/spark-warehouse
master	yarn-client
spark.app.id	application_1506539821041_0001
spark.app.name	Zeppelin
spark.driver.appUIAddress	http://10.132.0.13:4040
spark.driver.extraClassPath	::/usr/lib/zeppelin/local-repo/2CNJ6M6YM/*:/usr/lib/zeppelin/interpreter/spark/*::/usr/lib/zeppeli 0.6.2.jar:/usr/lib/zeppelin/interpreter/spark/zeppelin-spark_2.10-0.6.2.jar
spark.driver.extraJavaOptions	-Dfile.encoding=UTF-8 -Dlog4j.configuration=file:///etc/zeppelin/conf/log4j.properties -Dzeppelin.log.file=/var/log/zeppelin/zeppelin-interpreter-spark-zeppelin-cluster-dataengineer-n
spark.driver.host	10.132.0.13
spark.driver.maxResultSize	960m
spark.driver.memory	1920m
spark.driver.port	42973
spark.dynamicAllocation.enabled	true
spark.dynamicAllocation.maxExecutors	10000
spark.dynamicAllocation.minExecutors	1
spark.eventLog.dir	hdfs://cluster-dataengineer-mark-m/user/spark/eventlog
spark.eventLog.enabled	true
spark.executor.cores	2
spark.executor.extraJavaOptions	-Xbootclasspath/p:/usr/local/share/google/alpn/alpn-boot-8.1.7.v20160121.jar