Scala - Python - Spark

1) Accumulators

Scala

```
// File: AccumulatorExample.scala
import org.apache.spark.sql.SparkSession

object AccumulatorExample {
  def main(args: Array[String]): Unit = {
    val spark =
SparkSession.builder.appName("AccumulatorExample").master("l
ocal[*]").getOrCreate()
    val sc = spark.sparkContext

    val rdd = sc.parallelize(1 to 10)
    val accum = sc.longAccumulator("sumAccumulator")

    // Use accumulator inside an action
    rdd.foreach(x => accum.add(x))
    println(s"Accumulator sum = ${accum.value}") // Should
print 55

    spark.stop()
  }
}
```

Run with `spark-submit` after packaging (or run from sbt/IDE).

PySpark

```python
# File: accumulator_example.py
from pyspark.sql import SparkSession

spark =
SparkSession.builder.appName("AccumulatorExample").master("l
ocal[*]").getOrCreate()
sc = spark.sparkContext

rdd = sc.parallelize(range(1, 11))
accum = sc.accumulator(0)  # integer accumulator

def add_val(x):
    global accum
    accum += x

rdd.foreach(add_val)
print("Accumulator sum =", accum.value)  # Should print 55

spark.stop()
```

Run: `spark-submit accumulator_example.py`

---

2) Broadcast Variables

Scala

```scala
// File: BroadcastExample.scala
import org.apache.spark.sql.SparkSession

object BroadcastExample {
  def main(args: Array[String]): Unit = {
    val spark =
SparkSession.builder.appName("BroadcastExample").master("loc
al[*]").getOrCreate()
    val sc = spark.sparkContext

    val lookup = Map(1 -> "one", 2 -> "two", 3 -> "three")
    val bc = sc.broadcast(lookup)

    val rdd = sc.parallelize(Seq(1,2,3,2,1,4))
    val mapped = rdd.map(x => (x, bc.value.getOrElse(x,
"unknown")))
    mapped.collect().foreach(println)

    spark.stop()
  }
}
```

PySpark

```
# File: broadcast_example.py
from pyspark.sql import SparkSession

spark =
SparkSession.builder.appName("BroadcastExample").master("loc
al[*]").getOrCreate()
sc = spark.sparkContext

lookup = {1: "one", 2: "two", 3: "three"}
bc = sc.broadcast(lookup)

rdd = sc.parallelize([1,2,3,2,1,4])
mapped = rdd.map(lambda x: (x, bc.value.get(x, "unknown")))
print(mapped.collect())
```

```
spark.stop()
```

---

3) Piping to External Programs (`RDD.pipe`)

> Useful to call small external scripts (grep, awk, custom binaries). Works per-partition.

Scala

```
// File: PipeExample.scala
import org.apache.spark.sql.SparkSession

object PipeExample {
  def main(args: Array[String]): Unit = {
    val spark =
SparkSession.builder.appName("PipeExample").master("local[*]
").getOrCreate()
    val sc = spark.sparkContext

    val rdd = sc.parallelize(Seq("apple", "banana",
"cranberry", "banana"))
    // Example: use `tr` to uppercase (UNIX command) or
`grep`. Needs `tr` available on executor nodes.
    val piped = rdd.pipe("tr '[:lower:]' '[:upper:]'") //
each partition sent to tr
    piped.collect().foreach(println)

    spark.stop()
  }
}
```

PySpark

```
# File: pipe_example.py
from pyspark.sql import SparkSession

spark =
SparkSession.builder.appName("PipeExample").master("local[*]
").getOrCreate()
sc = spark.sparkContext

rdd = sc.parallelize(["apple", "banana", "cranberry",
"banana"])
# Use `tr` to uppercase (system must have tr). Alternatively
use a custom script path.
piped = rdd.pipe("tr '[:lower:]' '[:upper:]'")
print(piped.collect())

spark.stop()
```

Note: `pipe()` sends the lines of each partition to the stdin of the command. For production, prefer native Spark transformations (avoid external programs unless required).

---

4) Numeric RDD Operations (sum, mean, stdev, count, reduce)

Scala

```
// File: NumericOps.scala
import org.apache.spark.sql.SparkSession
import org.apache.spark.util.StatCounter

object NumericOps {
  def main(args: Array[String]): Unit = {
    val spark =
SparkSession.builder.appName("NumericOps").master("local[*]"
).getOrCreate()
    val sc = spark.sparkContext

    val rdd = sc.parallelize(Seq(1.0, 2.0, 3.0, 4.0, 5.0))
    println("sum: " + rdd.sum())
    println("count: " + rdd.count())
    println("mean: " + (rdd.sum() / rdd.count()))

    // StatCounter for many stats
    val stats = rdd.stats()
    println(s"mean=${stats.mean}, stdev=${stats.stdev},
max=${stats.max}, min=${stats.min}")

    spark.stop()
  }
}
```

PySpark

```
# File: numeric_ops.py
from pyspark.sql import SparkSession
from pyspark import SparkContext

spark =
SparkSession.builder.appName("NumericOps").master("local[*]"
).getOrCreate()
sc = spark.sparkContext

rdd = sc.parallelize([1.0,2.0,3.0,4.0,5.0])
print("sum:", rdd.sum())
print("count:", rdd.count())
print("mean:", rdd.mean())
```

```
stats = rdd.stats()
print("mean:", stats.mean(), "stdev:", stats.stdev(),
"max:", stats.max(), "min:", stats.min())

spark.stop()
```

---

5) Spark Runtime — basic introspection (app id, job/stage info, UI url)

> You can inspect basic runtime metadata from the `SparkContext`. For deeper tracing use `SparkListener` or the Web UI.

Scala

```
// File: RuntimeInfo.scala
import org.apache.spark.sql.SparkSession

object RuntimeInfo {
  def main(args: Array[String]): Unit = {
    val spark =
SparkSession.builder.appName("RuntimeInfo").master("local[*]
").getOrCreate()
    val sc = spark.sparkContext

    println(s"Application ID: ${sc.applicationId}")
    // UI web url if available (local mode shows a URL)
    println(s"Spark UI: ${sc.uiWebUrl.getOrElse("N/A")}")

    val rdd = sc.parallelize(1 to 1000, 4).map(_ * 2)
    // Trigger a job to generate stages & tasks
    val s = rdd.filter(_ % 3 == 0).count()
    println(s"Result count: $s")

    // StatusTracker for simple job/stage info
    val status = sc.statusTracker
    val jobIds = status.getJobIdsForGroup("") // may return
empty if groups not used
    println("Known job IDs: " + jobIds.mkString(","))

    spark.stop()
  }
}
```

PySpark

```
# File: runtime_info.py
from pyspark.sql import SparkSession

spark =
SparkSession.builder.appName("RuntimeInfo").master("local[*]
").getOrCreate()
sc = spark.sparkContext

print("Application ID:", sc.applicationId)
print("Spark UI:", sc.uiWebUrl if sc.uiWebUrl is not None
else "N/A")

rdd = sc.parallelize(range(1,1001), 4).map(lambda x: x*2)
count = rdd.filter(lambda x: x % 3 == 0).count()
print("Result count:", count)

# Status tracker
status = sc.statusTracker()
print("Known Job IDs:", status.getJobIdsForGroup(None))  #
may return []
```

```
spark.stop()
```

Notes:

- In cluster modes, UI will be on the driver node; `sc.uiWebUrl` may be accessible from driver host.
- For production tracing, register a `SparkListener` to collect stage/task events.

---

6) Deploying Applications

Two short examples showing how to package/run Scala and Python apps.

Scala: Build + spark-submit (sbt + assembly)

1. Add `sbt-assembly` plugin to `project/plugins.sbt`:

```
addSbtPlugin("com.eed3si9n" % "sbt-assembly" % "2.1.5")
```

2. In `build.sbt` define `mainClass` and dependencies (your Spark version).
3. Build the fat JAR:

```
sbt clean assembly
# produces target/scala-2.12/your-app-assembly-<version>.jar
```

4. Submit to cluster:

```
spark-submit \
  --class com.yourorg.YourMainClass \
  --master yarn \                # or spark://, or k8s://
  --deploy-mode cluster \
  --num-executors 4 \
  --executor-memory 4G \
  target/scala-2.12/your-app-assembly-1.0.jar arg1 arg2
```

Python: spark-submit

Create `my_app.py` (e.g., any of the PySpark examples above). Then:

```
spark-submit \
  --master yarn \
  --deploy-mode cluster \
  --conf spark.executor.memory=4g \
  --conf spark.executor.cores=2 \
  my_app.py arg1 arg2
```