

Spark Inception

Exploiting the Spark REPL to build Streaming Notebooks

with Scott Haines



< This Guy: Scott Haines: Spark Community Member, Databricks Beacon, Teacher, Author

< Experience: Startups, Yahoo, Twilio, Nike



Modern Data Engineering with Apache Spark

Apress*

Today's Material: Collection of experimental content from my book.

" I like data, hot sauce, and distributed systems" - Scott Haines

Plan of Attack

- 1. What is a Notebook Environment?
- 2. Building the Streaming Notebook Application
- 3. Demo Time...
- 4. Questions & Source Code

"you've all probably used notebooks, let's dive deeper into how they work and build our own"

Inception: The Point in which something begins its course or existence. A Start



Basics: The Science behind the Notebook Environment

What is the difference between Hypothesis & Law?

A law is a universal truth predicated on deterministic results given the same set of inputs. A hypothesis is a theory based on initial assumptions.

The notebook environment is the modern scientific notebook and facilitates the simple sharing of codified ideas and predictable results.

Popular Notebook Environments: What are they?



☆ ☆ ☆ î î ▶ ■ ♂ ▶ Markdown ∨

 [1]:
 from matplotlib import pyplot as plt import numpy as np

Generate 100 random data points along 3 dimensions
x, y, scale = np.random.randn(3, 100)
fig, ax = plt.subplots()

Map each onto a scatterplot we'll create with Matplotlib ax.scatter(x=x, y=y, c=scale, s=np.abs(scale)*500) ax.set(title="Some random data, created with JupyterLab!") plt.show()



Cmd 25

1 %: 2 S I 3 I 4 (sqt ELECT origin, FROM departure GROUP BY origi ORDER BY num_f	destination, c delays n, destination lights DESC	ount(*) as num_	flights	import org.apc import org.apc import org.apc import org.apc import org.apc
▶ (2)	Spark Jobs				val netflixCat val netflixCat
Tabl	e +				val contentWit .format("org .schema(net)
	origin 🔺	destination 🔺	num_flights 🔺		.option("key
1	SFO	LAX	3232		.load()
2	LAX	SFO	3198		contentWithCa
3	LAS	LAX	3016		usl setsemul
4	LAX	LAS	2964		.setHandleI
5	JFK	LAX	2720		.setInputCo
6	LAX	JFK	2719		
7	ΔΤΙ	I GA	2501		val ratingInd .setHandleI
:	Truncated resul	ts, showing first 1	1,000 rows. ∽ 0	.88 seconds runtime	.setInputCo
A s	QL cell result sto	ored as PvSpark d	ata frame soldf	Learn more	val fittedCate



import org.apache.spark.ml.feature.{IndexToString, StringIndexer, OneHotEncoderEstimator, VectorAssembler}
import org.apache.spark.ml.Pipeline

import org.apache.spark.ml.PipelineModel

import org.apache.spark.ml.clustering.KMeans import org.apache.spark.ml.evaluation.ClusteringEvaluator

val netflixCategoryDataWithRatingSchema = "`show_id' BIGINT,`category` STRING, `rating` STRING"
val netflixCategoryStructSchema = StructType.fromDDL(netflixCategoryDataWithRatingSchema)

val contentWithCategories = spark.read

- .format("org.apache.spark.sql.redis")
- .schema(netflixCategoryStructSchema)
- .option("key.column", "show_id")
 .option("table", "netflix_category_rating")
- .load()

ntentWithCategories.printSchema

- val categoryIndexer = new StringIndexer()
- .setHandleInvalid("keep") // options are keep, skip, error
- .setInputCol("category")
 .setOutputCol("category_index")
- val ratingIndexer = new StringIndexer()
- .setHandleInvalid("keep")
- .setInputCol("rating")
 .setOutputCol("rating_index")
- .secoutputtoi("rating_index")
- val fittedCategoryIndexer = categoryIndexer.fit(contentWithCategories.select("category").distinct())
 val fittedRatingIndexer = ratingIndexer.fit(contentWithCategories.select("rating").distinct())



Notebook Environments: Why they are Awesome?

In a Nutshell:

Notebook Environments enable you, the engineer, to focus on **"what your software 'should' do"** vs "all the other stuff"





Notebook Environments: Why they are Awesome?

In a Nutshell:

Notebook Environments enable you to **quickly test ideas** (hypothesis) *interactively*. Each code block (*paragraph*) is stitched together to *procedurally generate* (code), that when run inseries, produces reproducible results (output).

"on-the-fly data (engineering / science) pipelines"

Twitter: <a>@newfront Source Code: https://bit.ly/spark-inception

	me?					
dc	atabri	icks				
25	sql	destination a	count(+) as num fl	i - h -		
▶ (2) Tab	FROM departure GROUP BY orig ORDER BY num_ Spark Jobs	e_delays in, destination flights DESC		Ignts		
▶ (2) Tab	FROM departure GROUP BY orig ORDER BY num_ Spark Jobs	destination	num flights	Ignts		
▶ (2) Tab	FROM departure GROUP BY orig ORDER BY num_ Spark Jobs	destination A	num_flights 🔺	Ignts	_	
▶ (2) Tab	FROM departur GROUP BY orig ORDER BY num_ Spark Jobs	e_delays in, destination flights DESC destination LAX SFO	num_flights ▲ 3232 3198	Ignts		
► (2) Tab 1 2 3	FROM departur GROUP BY orig ORDER BY num_ Spark Jobs ie + origin SFO LAX LAS	e_delays in, destination flights DESC destination LAX SFO LAX	num_flights	Ignts		
2 S 3 ↓ 5 (2) Tab	FROM departur GROUP BY orig ORDER BY num_ Spark Jobs le + origin SFO LAX LAS LAX	destination destination LAX SFO LAS	num_flights	Ignts		
2 S 3 4 5 1 2 3 4 5	RECEIPTING AND	destination destination LAX SFO LAX LAX LAX	num_flights 3232 3198 3016 2964 2720	Ignts		
2 S 3 F 5 F 1 2 3 4 5 6	RECEIPTING departur GROUP BY orig ORDER BY num_ Spark Jobs le + origin SFO LAX LAX JFK LAX	destination destination LAX SFO LAX LAX LAX LAX LAX LAX LAX	num_flights 3232 3198 3016 2964 2720 27219	Ignts		
2 S 3 ↓ 5 (2) Tab 1 2 3 4 5 6 6 7	RECEI OFIGIN. GROUP BY orig ORDER BY num_ Spark Jobs le + origin ▲ SFO LAX LAS LAX LAS LAX JFK LAX ATI	destination destin	num_flights	Ignts		
1 1 2 S 3 4 5 6 7	RECEI OFIGINI, GROUP BY orig ORDER BY num_ Spark Jobs le + origin ▲ SFO LAX LAS LAX LAS LAX LAX JFK LAX ATI Truncated recu	destination destin	num_flights	8 seconds runtime		
1 2 3 4 5 6 7 :	RECEIPANG departur GROUP BY orig ORDER BY num_ Spark Jobs le + origin SFO LAX LAS LAX LAS LAX JFK LAX ATI Truncated resu	destination destination flights DESC	num_flights ▲ 3232 3198 3016 2964 2720 2719 2501 1,000 rows. ~ 0.8	8 seconds runtime		

Basics: The Anatomy of a Digital Notebook?

The Notebook:

Stores your software (code) and notes (markdown / simple text) in a series of blocks (paragraphs or cells)



Basics: The Anatomy of a Digital Notebook?

Notebook > Paragraph(s):

Each paragraph can be modified, compiled, and run independently (async) or the collection (notebook) can be run as a pipeline.



Basics: The Anatomy of a Digital Notebook?

Recap:

Notebook environments help you move fast, and test and share ideas without the trouble of "local developer environment" pain.



Part 2: Building the Streaming Notebook Application

Under the Hood:

Learn how to remote control the Spark REPL (think spark-shell) using streaming remote procedure calls (RPC).



What is the Spark REPL?



```
Spark REPL Source Code
```

Twitter: <a>@newfront

/** Print a welcome message */
override def printWelcome(): Unit = {
 import org.apache.spark.SPARK_VERSION
 echo("""Welcome to









Architecture: RPC Command: NetworkCommand

```
case class NetworkCommand(
  notebookld: String,
  paragraphId: String,
  command: String,
  requestId: String,
  userId: Option[String] = Some("nobody")
```

< This is the metadata encapsulating a notebook, the paragraph, the command to be evaluated, and more goodies.

com:coffeeco:notebooks:v1:notebook1:rpc

< This is a Redis Stream for the transport of our NetworkCommand's

Architecture: RPC Command: NotebookExecutionDetails

case class NotebookExecutionDetails(
 notebookId: String,
 paragraphId: String,
 command: String,
 requestId: String,
 userId: Option[String],
 commandStatus: String,
 result: String)

< This is the metadata encapsulating the complete execution of a remote notebook paragraph run

com:coffeeco:notebooks:v1:notebook1:results < This is a Redis location for the

< This is a Redis location for the collection of our processed commands

Spark Inception App: RPC Processing Flow

```
override def runApp(): StreamingQuery = {
    logger.info(s"run.app.called")
    import sparkSession.implicits._
```

```
// the inception pipeline
outputStream(
    NetworkCommandProcessor(sparkSession).process(
        inputStream.load().as[NetworkCommand]
    ).writeStream
).foreachBatch((ds: Dataset[NetworkCommand], batchId: Long) => processBatch(ds, batchId)
).start()
```

SparkInceptionControllerApp.runApp()



processBatch (part 1: processCommand)

For each microbatch:

- Collect the RPC commands (bring to driver)
- 2. Evaluate each command.
- 3. Repeat forever...

SparkInceptionControllerApp.processBatch

```
def processBatch(ds: Dataset[NetworkCommand], batchId: Long): Unit = {
    import sparkSession.implicits._
    // Collect all of the Distributed Commands and bring down to the Driver
    val localResults = ds.collect().map { networkCommand =>
        // this is running on the driver now (not the executors)
    val res = sparkRemoteSession.processCommand(networkCommand)
        // wrap the execution details so we can write the results to redis
        NotebookExecutionDetails(
            networkCommand.notebookId,
            networkCommand.command,
            networkCommand.requestId,
            networkCommand.userId,
            res.commandStatus,
            res.consoleOutput
        )
    }.toSeg
```

Twitter: <u>@newfront</u>

Source Code: https://bit.ly/spark-inception



processCommand:

From the Spark Driver:

- 1. Evaluate: should we run?
- Redirect the commands (%spark or %sql)
- 3. Respond (return results)

SparkRemoteSession.processCommand

```
def processCommand(cmd: NetworkCommand): NetworkCommandResult = {
 initialize()
 val user = cmd.userId.getOrElse("nobody")
 val parsed = cmd.parse()
 if (parsed._1 != Command.UnsupportedCommand && authCheck(user)) {
    logger.debug(s"security.gate.passed")
    val results = Console.withOut(consolePrintStream) {
      System.setOut(Console.out)
     parsed._1 match {
       case SparkCommand =>(processSparkScala(parsed._2))
        case SparkSQLCommand => processSparkSQL(parsed._2)
        case _ => (Status.Failure, s"${cmd.command} is not supported")
    System.setOut(initialConsoleOutputStream)
    NetworkCommandResult(
     requestId = cmd.requestId,
      commandStatus = results._1,
     consoleOutput = results._2
   else NetworkCommandResult(cmd.requestId, "Failure", s"$user is not authorized")
```

Twitter: <u>@newfront</u>

Source Code: https://bit.ly/spark-inception



processSparkScala:

- 1. Interpret (eg: use the REPL)
- 2. return results

SparkRemoteSession.processSparkScala

- def processSparkScala(cmd: String): (String, String) = {
 val result = sparkILoop.interpret(cmd, synthetic = true)
 val consoleOutput = readOutput()
 (result.toString, consoleOutput.mkString("\n"))
- ់

Source Code: https://bit.ly/spark-inception

Twitter: <a>@newfront



processSparkSQL:

1. Try to evaluate the SQL

(unsafe sure!)

2. Capture results

Twitter: <u>@newfront</u>

SparkRemoteSession.processSparkSQL

def processSparkSQL(cmd: String): (String, String) = {

// note: In the case where you want delete protection for tables
// or want to add specific limits (like limit 10 for open queries)
// then you can parse the cmd string and add magic

Try(app.sparkSession.sql(cmd)) match { case Success(df: DataFrame) => (Status.Success, df.toJSON.collect().toSeq.mkString("\n")) case Failure(ex: Exception) => ex.printStackTrace(consolePrintStream) (Status.Failure, readOutput().mkString("\n")) case Failure(thr: Throwable) => thr.printStackTrace(consolePrintStream) (Status.Failure, readOutput().mkString("\n")) case _ => (Status.Failure, "Something went wrong")

Source Code: https://bit.ly/spark-inception



processBatch (part 2: results to redis)

After evaluating the NetworkCommands:

- 1. Create a Results Dataset
- 2. Write to our Redis output table
- 3. Repeat forever...

SparkInceptionControllerApp.processBatch

```
def processBatch(ds: Dataset[NetworkCommand], batchId: Long): Unit = {
    import sparkSession.implicits._
    // Collect all of the Distributed Commands and bring down to the Driver
    val localResults = ds.collect().map { networkCommand =>
        // this is running on the driver now (not the executors)
    val res = sparkRemoteSession.processCommand(retworkCommand)
        // wrap the execution details so we can write the results to redis
        NotebookExecutionDetails(
            networkCommand_neregraphId
        // generate a new dataframe and then write back to redis
    val forRedis = sparkSession.createDataset[NotebookExecutionDetails](localResults)
    forRedis
    .write
    .format( source = "org.apache.spark.sql.redis")
    .options(sparkSession.sparkContext
    }
}
```

```
.getConf
```

.getAllWithPrefix(appConfig.SinkStreamOptions).toMap[String, String])
.mode(SaveMode.Append)

```
.save()
```

Twitter: <a>@newfront

Source Code: https://bit.ly/spark-inception

Inception: The Point in which something begins its course or existence.

Part 3: Demo Time!



Thank You. Thanks. Now some Q&A

Github: <u>https://bit.ly/spark-inception</u> Medium: <u>https://medium.com/@newfrontcreative</u> LinkedIn: <u>https://www.linkedin.com/in/scotthaines/</u>

Docker: https://dockr.ly/30uSom2

