Spark Inception

Exploiting the Spark REPL to build Streaming Notebooks

with **Scott Haines**

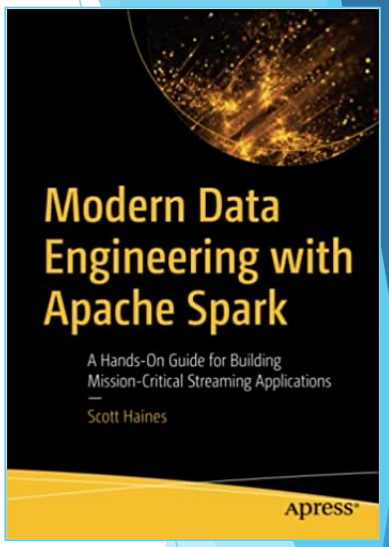
Twitter: @newfront

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

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

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

< Experience: Startups, Yahoo, Twilio, Nike



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

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

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"

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

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?

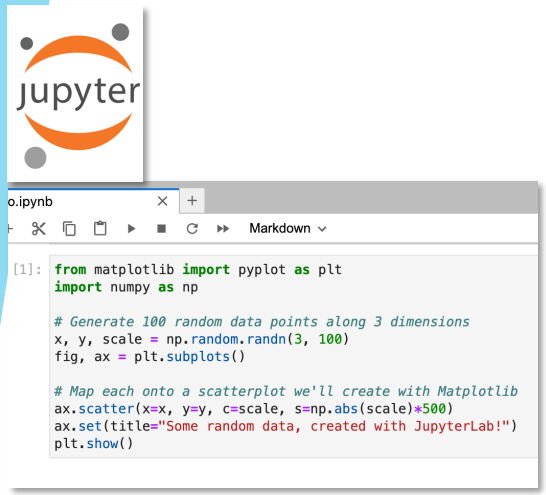
A hypothesis is a theory based on initial assumptions.

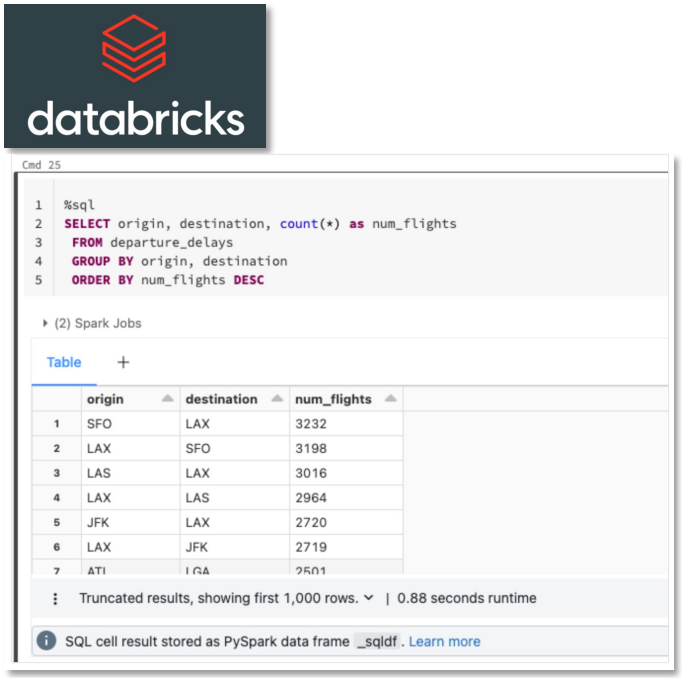
A law is a universal truth predicated on deterministic results given the same set of inputs.

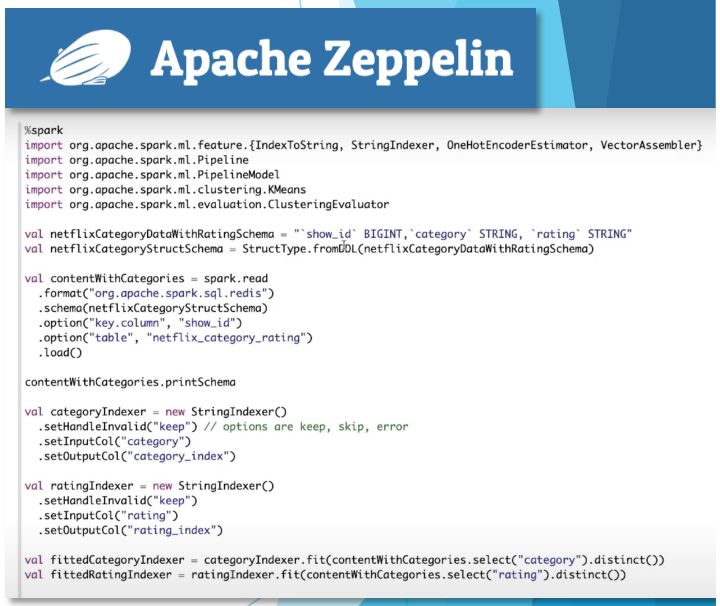
The notebook environment is the modern scientific notebook and facilitates the simple sharing of codified ideas and predictable results.

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

Popular Notebook Environments: What are they?





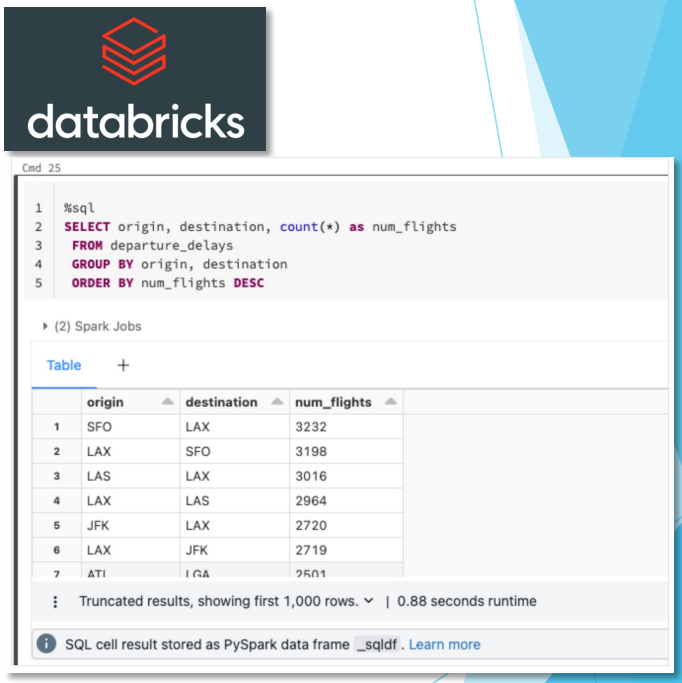


Twitter: @newfront Source Code: https://bit.ly/spark-inception

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”

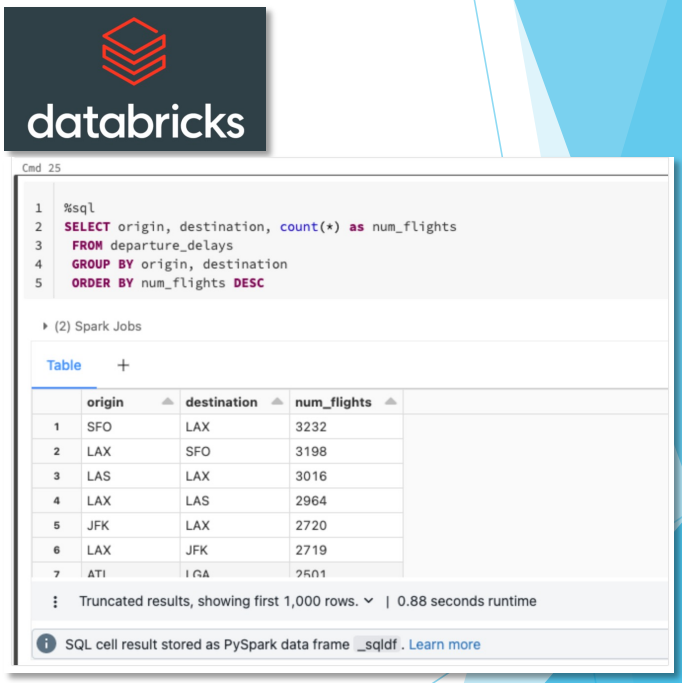


Twitter: @newfront Source Code: https://bit.ly/spark-inception

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 in-series, produces reproducible results (output).



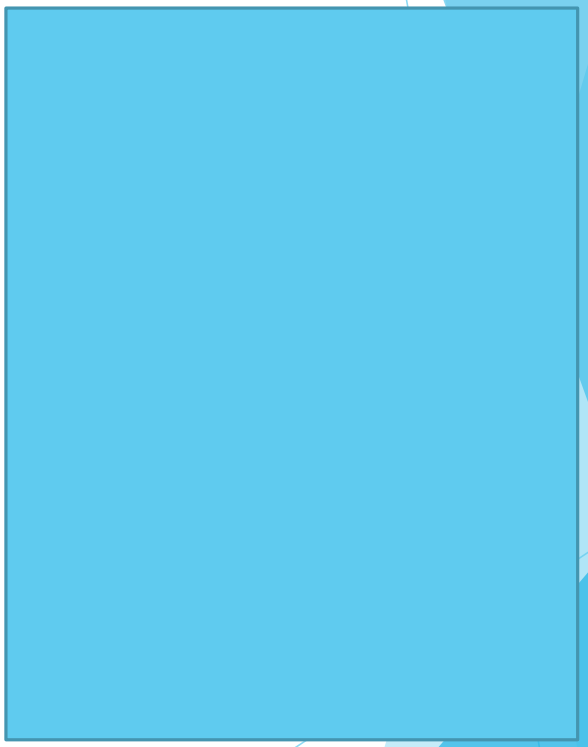
"on-the-fly data (engineering / science) pipelines"

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

Basics: The Anatomy of a Digital Notebook?

The Notebook:

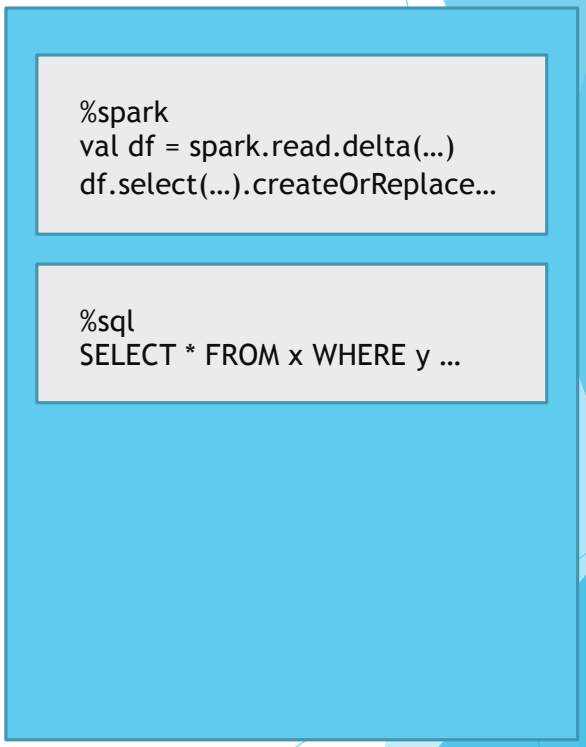
Stores your software (code) and notes (markdown / simple text) in a series of blocks (paragraphs or cells)



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.

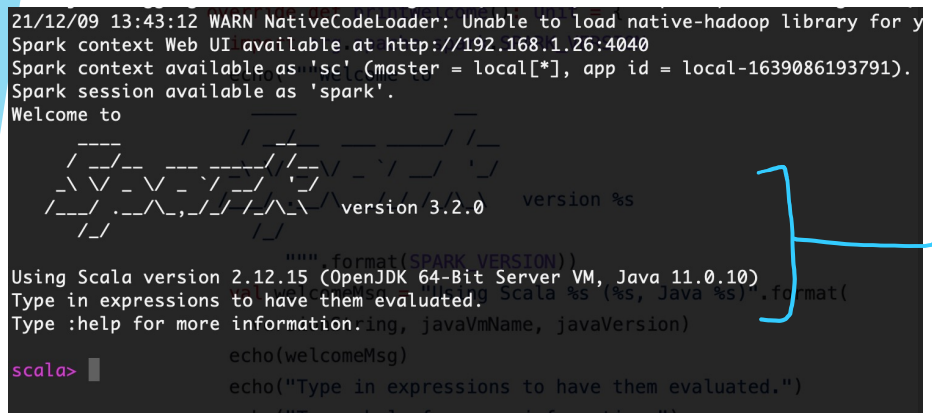
Part 2: Building the Streaming Notebook Application

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_HOME/bin/spark-shell
```

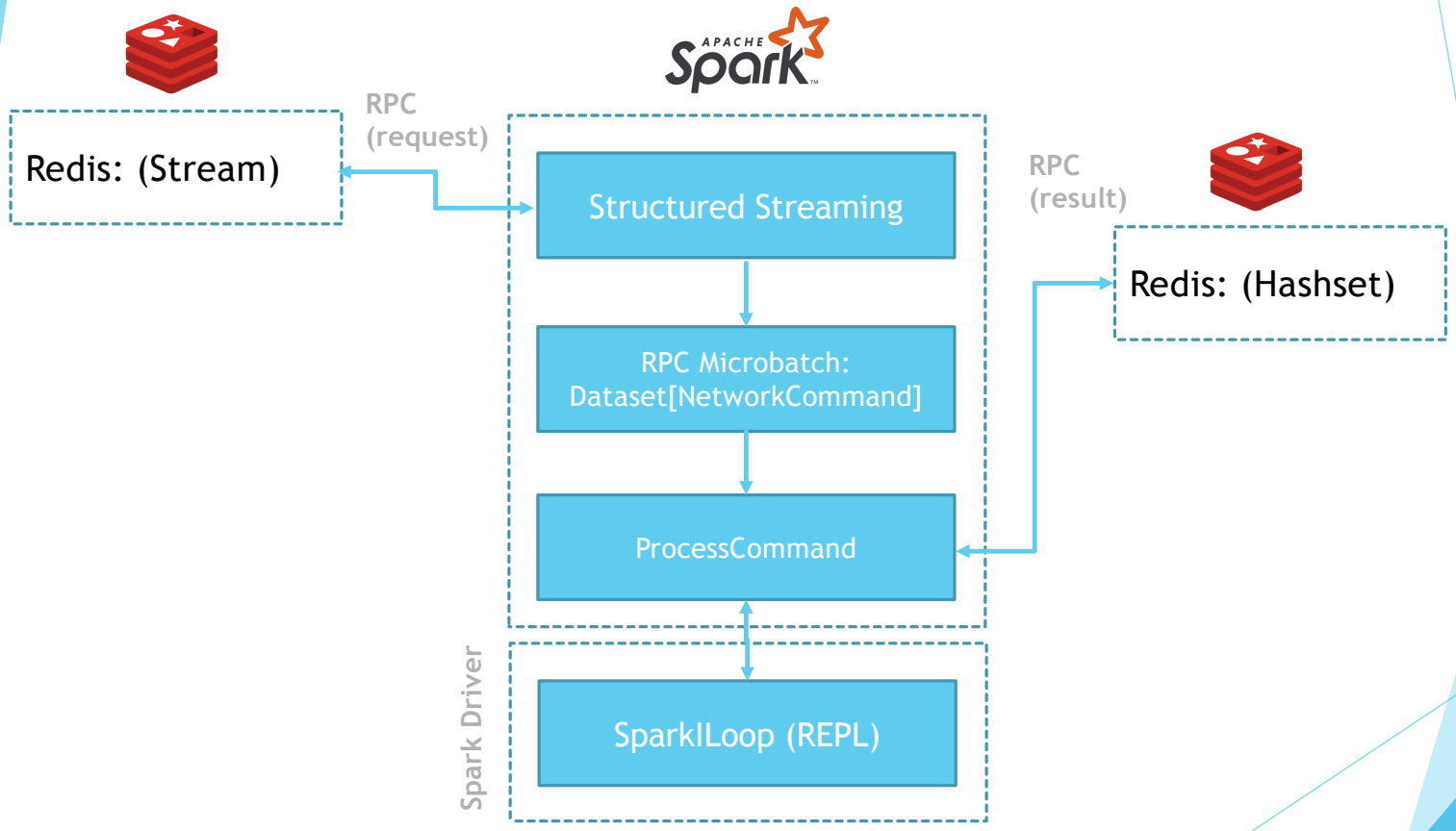


[Spark REPL Source Code]

```scala
/** Print a welcome message */
override def printWelcome(): Unit = {
  import org.apache.spark.SPARK_VERSION
  echo("""Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version %s
      /_/
         """.format(SPARK_VERSION))
  val welcomeMsg = "Using Scala %s (%s, Java %s)".format(
    versionString, javaVmName, javaVersion)
  echo(welcomeMsg)
  echo("Type in expressions to have them evaluated.")
  echo("Type :help for more information.")
}
```

Twitter: @newfront

Architecture: Overview



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

Architecture: RPC Command: NetworkCommand

```
case class NetworkCommand(
  notebookId: 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

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

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 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()

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

Spark Inception App: Command Processing

processBatch (part 1: processCommand)

For each microbatch:

1. **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.paragraphId,
      networkCommand.command,
      networkCommand.requestId,
      networkCommand.userId,
      res.commandStatus,
      res.consoleOutput
    )
  }.toSeq
```


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

Spark Inception App: Command Processing

processCommand:

From the Spark Driver:

1. Evaluate: should we run?
2. 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")
    // evaluate the command
    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")
}
```


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

Spark Inception App: Command Processing

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

Spark Inception App: Command Processing

processSparkSQL:

1. Try to evaluate the SQL

(unsafe sure!)

2. Capture results

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

Spark Inception App: Command Processing

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(networkCommand)
    // wrap the execution details so we can write the results to redis
    NotebookExecutionDetails(
      networkCommand.notebookId,
      networkCommand.paragraphId
```

```
  // 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: @newfront

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

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

Part 3: Demo Time!

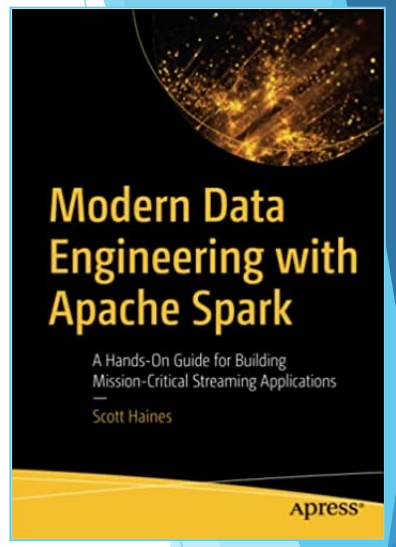
Thank You. Thanks. Now some Q&A

Github: https://bit.ly/spark-inception

Medium: https://medium.com/@newfrontcreative

LinkedIn: https://www.linkedin.com/in/scotthaines/

Docker: https://dockr.ly/3OuSom2



Learn at your own Pace