



Behavior-Driven Development (BDD) Testing with Apache Spark

Kevin Mellott

Technical Team Lead, Data and Analytics

Zachary Nanfelt

Software Engineer, Data and Analytics



Who is FIS Global?

- We're FIS Digital Finance, Mobile Data and Analytics
- One of the **largest global** FinTech companies
- Customers are banks and credit unions
- Ecosystem of products and services **built around core banking**

Data Wrangling

- 1 ETL is still a thing
- 2 The way we do it varies quite a bit
- 3 When we do it also varies
- 4 The why we do it, that's easy

Questions on what we are doing?

Can you prove that the data **transformed** correctly?

Is unit testing **understandable**?

Did we cover all **scenarios**?

Was **acceptance criteria** met?

Are we able to do all this testing complexity in a reasonable **timeframe**?

What is BDD?

- An extension of Test Driven Development
- Deliberate shared, ubiquitous language
- Automated acceptance tests written as *Examples* that anyone can read
- Living documentation – Documentation others can read about code is updated
- More agile to have good documentation as an output to development than making documentation as input
- Enable more Team Members to Participate in the development process

Core Problem of Data Transformation

- It is **really hard** to prove data transformed correctly in a **normal pipeline aka *Batch-Oriented***
- The **traditional** way has been to push data through the system and then **query it out**
- **Apache Spark** can accelerate not only the **speed you transform**, but the speed in which you can **validate** transformations
 - We can switch from Batch Oriented to Streaming

Spark is our favorite hammer



=



Beautiful Baby

Super Widget Scenario

- Our app servers log everything in **Epoch Time (Unix)** from mobile app clients **all over the world**
- Users seem incapable of computing this mentally and want it to appear in their own **timezone**
- *Crazy*, but some of these guys are **remote** and in **different timezones**

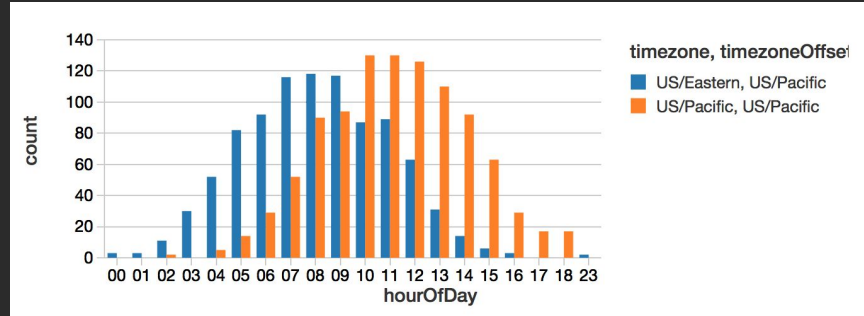
Extraction Code

```
1 package com.fis.mobile.examples
2
3 import ...
4
5
6 class ExtractionClassV1(sparkSession: SparkSession) {
7     val TIMESTAMP_FORMAT = "yyyy-MM-dd HH:mm:ss"
8
9
10    def RunExtractJob(sourceFilePath: String, destinationFilePath: String, timezoneOffset: String): Unit = {
11        val sourceDataFrame: DataFrame = GetJsonDataFrame(sourceFilePath)
12        val extractedDataFrame: DataFrame = ExtractDataFrame(sourceDataFrame, timezoneOffset)
13        SaveJsonDataFrame(extractedDataFrame, destinationFilePath)
14    }
15
16    def GetExtractedJsonDataFrame(filePath: String, timezoneOffset: String): DataFrame = {
17        ExtractDataFrame(GetJsonDataFrame(filePath), timezoneOffset)
18    }
19
20    private def GetJsonDataFrame(filePath: String): DataFrame = {
21        sparkSession.read.json(filePath)
22    }
23
24    private def ExtractDataFrame(dataFrame: DataFrame, timezoneOffset: String): DataFrame = {
25        import sparkSession.implicitly._
26
27        dataFrame
28            .withColumn("timezoneOffset", lit(timezoneOffset))
29            .withColumn("timestampGmt", from_unixtime($"unixTimestamp"))
30            .withColumn("timestampLtz",
31                date_format(from_utc_timestamp(from_unixtime($"unixTimestamp"), timezoneOffset), TIMESTAMP_FORMAT))
32    }
33
34    private def SaveJsonDataFrame(dataFrame: DataFrame, filePath: String): Unit = {
35        dataFrame.write.json(filePath)
36    }
37 }
38
```

SQL Validation Test Code

```
SELECT TIMEZONE_OFFSET,  
       TIMESTAMP_GMT,  
       TIMESTAMP_LTZ  
FROM  
       APPSTORE_REVIEWS  
WHERE  
       TIMEZONE_OFFSET <> 0  
LIMIT 10;
```

Cucumber Test



```
1 @Extraction @TempFileCleanup @ApacheSpark
2 Feature: Json Logs Extract Process V1
3
4 Background: general system setup
5   Given the system is in UTC time
6
7 @V1
8 Scenario: Basic extraction of Epoch time into readable local time zones specified by person doing extraction
9   Given there is a file "srcFolder/example.json" with the following lines:
10     | {"logId":1, "unixTimestamp":1459482142, "timezone":"US/Pacific"} |
11     | {"logId":2, "unixTimestamp":1459482142, "timezone":"US/Eastern"} |
12   When the method RunExtractJobV1 gets called with
13     | SourceFolder | srcFolder/* |
14     | DestinationFolder | dstFolder |
15     | TimezoneOffset | US/Pacific |
16   Then there will be a "_SUCCESS" file in the "dstFolder" folder
17   And the folder "dstFolder" will have json files with exactly the following DataFrame rows:
18     | logId | unixTimestamp | timestampLtz | timezoneOffset |
19     | 1 | 1459482142 | 2016-03-31 20:42:22 | US/Pacific |
20     | 2 | 1459482142 | 2016-03-31 20:42:22 | US/Pacific |
21
```

Cucumber Test Version 2

```
1 @Extraction @TempFileCleanup @ApacheSpark @V2  
2 Feature: Json Logs Extract Process V2
```

```
3  
4 Background: general system setup  
5 Given the system is in UTC time
```

```
6  
7 Scenario: Basic extraction of Epoch time into readable local time zones of the devices  
8 Given there is a file "srcFolder2/example.json" with the following lines:
```

```
9 | {"logId":1, "unixTimestamp":1459482142, "timezone":"US/Pacific"} |  
10 | {"logId":2, "unixTimestamp":1459482142, "timezone":"US/Eastern"} |
```

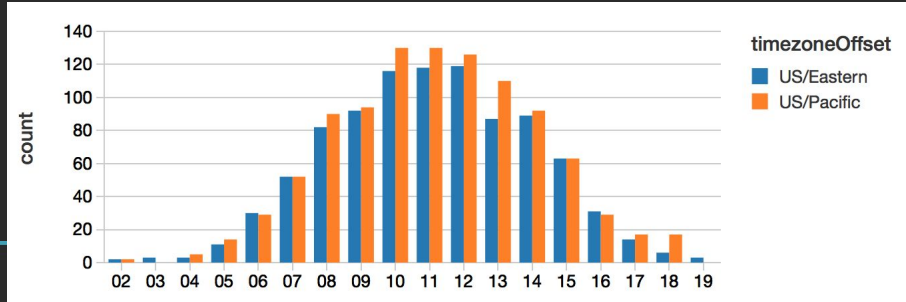
```
11 When the method RunExtractJobV2 gets called with
```

```
12 | SourceFolder | srcFolder2/* |  
13 | DestinationFolder | dstFolder2 |
```

```
14 Then there will be a "_SUCCESS" file in the "dstFolder2" folder
```

```
15 And the folder "dstFolder2" will have json files with exactly the following DataFrame rows:
```

```
16 | logId | unixTimestamp | timestampLtz | timezoneOffset |  
17 | 1 | 1459482142 | 2016-03-31 20:42:22 | US/Pacific |  
18 | 2 | 1459482142 | 2016-03-31 23:42:22 | US/Eastern |
```



Why this is so great

- Collaboration and Participation
- Thinking naturally begets better scenarios
- We are able to unify
 - Use Cases
 - We are using ETL...
 - All Projects



Given a BDD Presentation When it is late in the day And FIS is giving the talk Then get Excited!

- We define Features and Scenarios
Expressive Scenarios
- Given/When/Then
*Gherkin doesn't care how you use them
They just help with readability.*



Wait, there's more!

Step Definitions

- Step Definitions tell the *how* to do
The Feature file said *what* to do
- This the boundary of the programmer's
Domain and the business domain.
- It's not all snake oil, really...

Cucumber Step Definition Code

```
1 package com.fis.mobile.examples;
2
3 import ...
4
17
18 public class ExtractionStepDefinitions {
19     @Given("^the system is in UTC time$")
20     public void theSystemIsInGMTTime() throws Throwable {
21         TimeZone.setDefault(TimeZone.getTimeZone("UTC"));
22     }
23
24     @Given("^there is a file \"([^\"]*)\" with the following lines:$")
25     public void thereIsAFileWithTheFollowingLines(String propertiesPath, List<String> lines) throws Throwable {
26         File file = new File(Helpers.getTempTestPath(propertiesPath));
27         //noinspection ResultOfMethodCallIgnored
28         file.getParentFile().mkdirs();
29         PrintWriter writer = new PrintWriter(file.getAbsolutePath(), "UTF-8");
30         for (String str : lines) {
31             writer.println(str.trim());
32         }
33         writer.close();
34     }
35
36     @When("^the method (RunExtractJobV1|RunExtractJobV2) gets called with$")
37     public void theMethodRunExtractJobGetsCalledWith(String jobName, Map<String, String> arguments) throws Throwable {
38         if(jobName.compareTo( anotherString: "RunExtractJobV1")==0) {
39             new ExtractionClassV1(Helpers.testSparkSession).RunExtractJob(
40                 Helpers.getTempTestPath(arguments.get("SourceFolder")),
41                 Helpers.getTempTestPath(arguments.get("DestinationFolder")),
42                 arguments.get("TimezoneOffset"));
43         }
44         else if(jobName.compareTo( anotherString: "RunExtractJobV2")==0) {
45             new ExtractionClassV2(Helpers.testSparkSession).RunExtractJob(
46                 Helpers.getTempTestPath(arguments.get("SourceFolder")),
47                 Helpers.getTempTestPath(arguments.get("DestinationFolder")));
48         } else { throw new PendingException(); }
49     }
50 }
```

Cucumber Step Definition Code

```
51 @Then("^there will be a \"([^\"]*)\" file in the \"([^\"]*)\" folder$")
52 public void thereWillBeAFileInTheFolder(final String partialFileName, String destinationFolder) throws Throwable {
53     String[] filesInFolder = new File(Helpers.getTempTestPath(destinationFolder)).list();
54
55     assert filesInFolder != null;
56     Assert.assertTrue(Arrays.asList(filesInFolder).contains(partialFileName));
57 }
58
59 @Then("^the folder \"([^\"]*)\" will have json files with exactly the following DataFrame rows:$")
60 public void theFolderWillHaveJsonFilesWithTheFollowingDataFrameRows(
61     String destinationFolder, List<Map<String,String>> dataTable) throws Throwable {
62     Dataset<Row> actualDF = Helpers.testSparkSession.read().json(Helpers.getTempTestPath(destinationFolder)).cache();
63
64     Assert.assertEquals( message: "Number of rows in DataFrame", dataTable.size(), actualDF.count());
65
66     for(Map<String, String> expectedColumns : dataTable) {
67         Dataset<Row> actualRow = actualDF;
68
69         for (Map.Entry<String, String> expectedCell : expectedColumns.entrySet()) {
70             actualRow = actualRow.filter(String.format("%s <=> '%s'", expectedCell.getKey(), expectedCell.getValue()));
71         }
72
73         if (actualRow.count() != 1) {
74             System.out.println("ACTUAL (entire DataFrame:");
75             actualDF.show( truncate: false);
76
77             Assert.assertEquals(String.format("EXPECTED ROW: %s",
78                 expectedColumns.toString()), expected: 1, actualRow.count());
79         }
80     }
81 }
82 }
83 }
```

Cucumber Code

```
1 package com.fis.mobile.examples;
2
3 import ...
4
5
6
7 @RunWith(Cucumber.class)
8 @CucumberOptions(
9     glue = "com.fis.mobile.examples",
10    features = {"src/test/resources/Feature/"})
11
12 public class RunFeatures {
13
14 }
15
```

```
1 package com.fis.mobile.examples;
2
3 import ...
4
5
6
7
8
9
10
11 public class BeforeAfterHooks {
12     @Before("@TempFileCleanup")
13     @After("@TempFileCleanup")
14     public void CleanupTempFiles() throws IOException, InterruptedException {
15         File tempDirectory = new File(Helpers.getTempPath( relativePath: ""));
16         FileUtils.deleteDirectory(tempDirectory);
17     }
18
19     @Before("@ApacheSpark")
20     public void setup() {
21         if(Helpers.testSparkSession == null) {
22             Helpers.testSparkSession = SparkSession
23                 .builder()
24                 .master("local")
25                 .appName("cucumberSparkAppTestSession")
26                 .config("spark.driver.host", "127.0.0.1")
27                 .getOrCreate();
28         }
29     }
30 }
31
```

For these two guys, ETL wasn't hell,
it was target practice.



2 Kinds of People in this world...



THE GOOD THE BAD AND THE UGLY

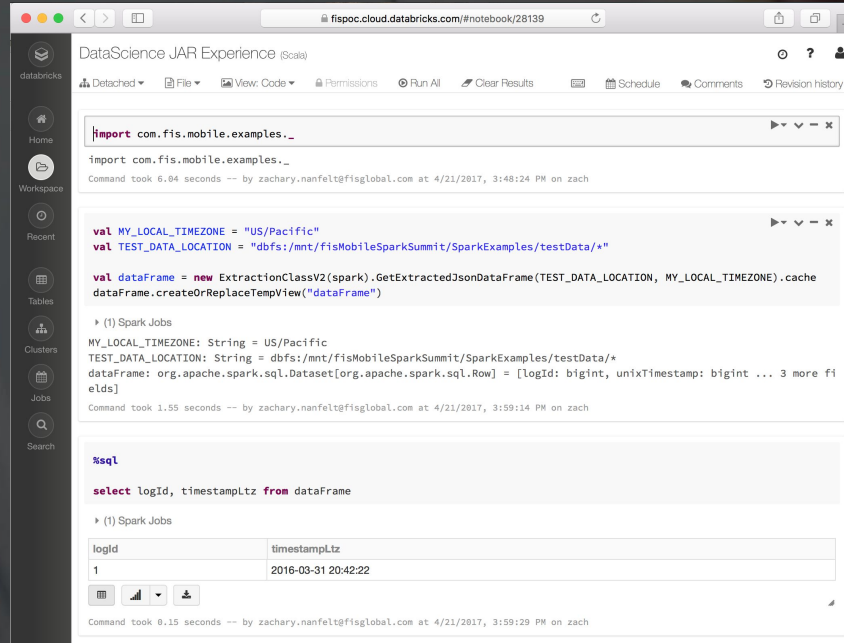
About 
cucumber

Enterprise Stuff

- You will notice we are sticking to **Eclipse/IntelliJ**
- Enterprises usually need to prove **Separation of Duties** and **Audit Trails**
- Most Data processing tasks *should* have an established process to ensure quality and correctness.
 - All Business have their own Custom Approach to Business Rules
 - Consolidating these transformations ensure quality. Allows QA Checking
- Notebooks:
 - It's really hard to enforce that consistency and correctness in notebooks, except by Compiling libraries.
 - Unifying Business Logic and Common Transformations removes the prep work.

Code

<https://dbc-39f78c99-dfb2.cloud.databricks.com/#notebook/28139>



The screenshot shows a Databricks notebook interface. The browser address bar displays `fsppoc.cloud.databricks.com/#notebook/28139`. The notebook title is "DataScience JAR Experience (Scala)". The interface includes a sidebar with navigation options like Home, Workspace, Recent, Tables, Clusters, Jobs, and Search. The main content area shows a code cell with the following Scala code:

```
import com.fis.mobile.examples._

import com.fis.mobile.examples._

val MY_LOCAL_TIMEZONE = "US/Pacific"
val TEST_DATA_LOCATION = "dbfs:/mnt/fisMobileSparkSummit/SparkExamples/testData/*"

val dataframe = new ExtractionClassV2(spark).GetExtractedJsonDataFrame(TEST_DATA_LOCATION, MY_LOCAL_TIMEZONE).cache
dataframe.createOrReplaceTempView("dataFrame")
```

Below the code, the execution results are displayed:

(1) Spark Jobs

MY_LOCAL_TIMEZONE: String = US/Pacific
TEST_DATA_LOCATION: String = dbfs:/mnt/fisMobileSparkSummit/SparkExamples/testData/*
dataFrame: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [logId: bigint, unixTimestamp: bigint ... 3 more fields]

Command took 6.04 seconds -- by zachary.nanfelt@fisglobal.com at 4/21/2017, 3:48:24 PM on zach

The next code cell contains SQL code:

```
%sql

select logId, timestampLtz from dataFrame
```

Execution results for the SQL cell:

(1) Spark Jobs

logId	timestampLtz
1	2016-03-31 20:42:22

Command took 0.15 seconds -- by zachary.nanfelt@fisglobal.com at 4/21/2017, 3:59:29 PM on zach

Tips and Tricks (Pretty Report)

- plugin = {"pretty", "html:target/cucumber"} isn't very pretty
- Use *cucumber-reports*

▼ @Extraction @TempFileCleanup @ApacheSpark Feature: Json Logs Extract Process

▼ Background: general system setup
Given the system is in UTC time

▼ @V1 Scenario: Basic extraction of Epoch time into readable local time zones
Given there is a file "srcFolder/example.json" with the following lines:

```
{"logId":1,"unixTimestamp":1459482142,"timezoneOffset":-6}  
{"logId":2,"unixTimestamp":1459482142,"timezoneOffset":-2}
```

When the method RunExtractJob gets called with

SourceFolder	srcFolder/*
DestinationFolder	dstFolder

Then there will be a "_SUCCESS" file in the "dstFolder" folder
And the folder "dstFolder" will have json files with exactly the following DataFrame rows:

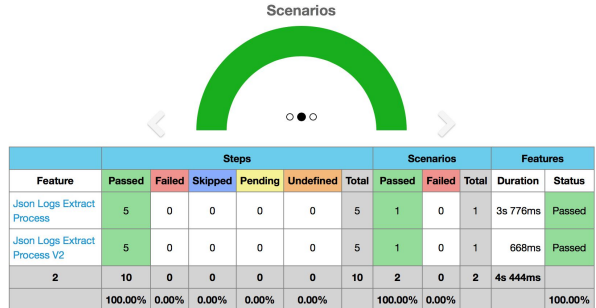
logId	unixTimestamp	timezoneOffset	timestampGmt	timestampLtz
1	1459482142	-6	2016-04-01 03:42:22	2016-03-31 21:42:22
2	1459482142	-2	2016-04-01 03:42:22	2016-04-01 01:42:22

Verses

Project	Number	Date
cucumber-jvm-example	1	30 May 2017, 22:43

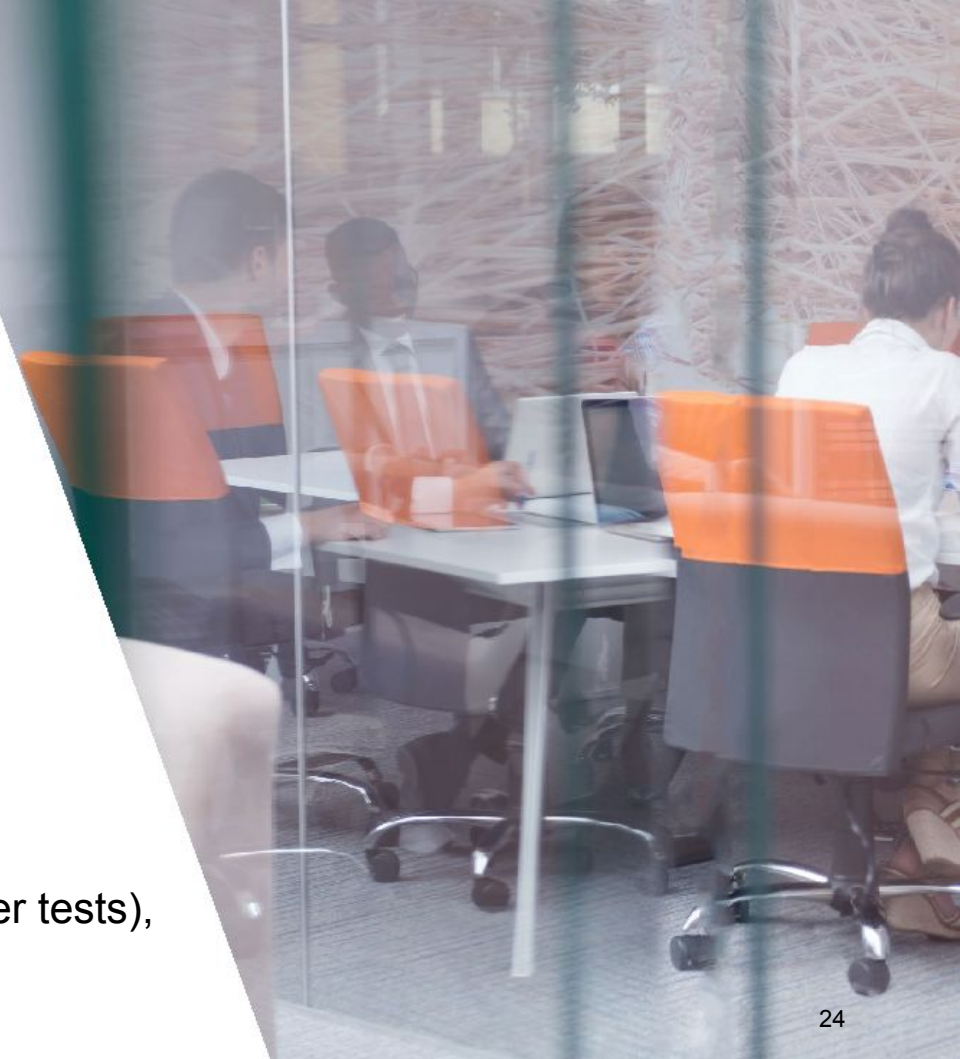
Features Statistics

The following graphs show passing and failing statistics for features



Tips and Tricks (miscellaneous)

- Java cuke > Scala cuke
 - IntelliJ integration, speed, etc...
- `.config("spark.driver.host", "127.0.0.1")`
 - Saves overall test execution time for each run
- Think hard about whether things should get tested at unit or component level
 - DAG takes longer to compute path on more complex DAGs (e.g. longer tests), but can provide more value



Tips and Tricks (Code Coverage for Scala Cuke)

The screenshot shows the 'Run/Debug Configurations' dialog in IntelliJ IDEA. The configuration is named 'Feature: extractTests'. Under the 'Code Coverage' tab, the 'Tracing' option is selected. The 'Packages and classes to record coverage data' section shows 'com.*' selected. The 'Coverage' window displays the following data:

Coverage RunFeatures

66% classes, 86% lines covered in package 'com.fis.mobile.examples'

Element	Class, %	Method, %	Line, %
ExtractionCl...	100% (1/1)	100% (6/6)	100% (12/12)
ExtractionCl...	100% (1/1)	100% (7/7)	100% (13/13)
sparkCucu...	0% (0/1)	0% (0/2)	0% (0/4)

Resources, Resources, Resources

- [Cucumber.io](https://cucumber.io)
- [cucumber-reporting](https://cucumber-reporting.com)
- Pragmatic Books
 - [Cucumber Book](#)
 - [Cucumber Recipes](#)
 - [Cucumber for Java](#)
- [Specification by Example](#)
- [Databricks Blog](#)





Thank you

Kevin Mellott
kevin.mellott@fisglobal.com

Zachary Nanfelt
zachary.nanfelt@fisglobal.com

