Behavior-Driven Development (BDD) Testing with Apache Spark

Aaron Colcord
Director of Engineering, 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**

Boilerplate Code

```scala
package com.fis.mobile.examples

import org.apache.spark.sql.SparkSession

object sparkCucumber {
  def main(args: Array[String]) {
    val sparkSession: SparkSession = SparkSession.builder.getOrCreate()

    new ExtractionClassV1(sparkSession)
      .RunExtractJob(sourceFilePath = args(0), destinationFilePath = args(1), timezoneOffset = args(3))
  }
}
```

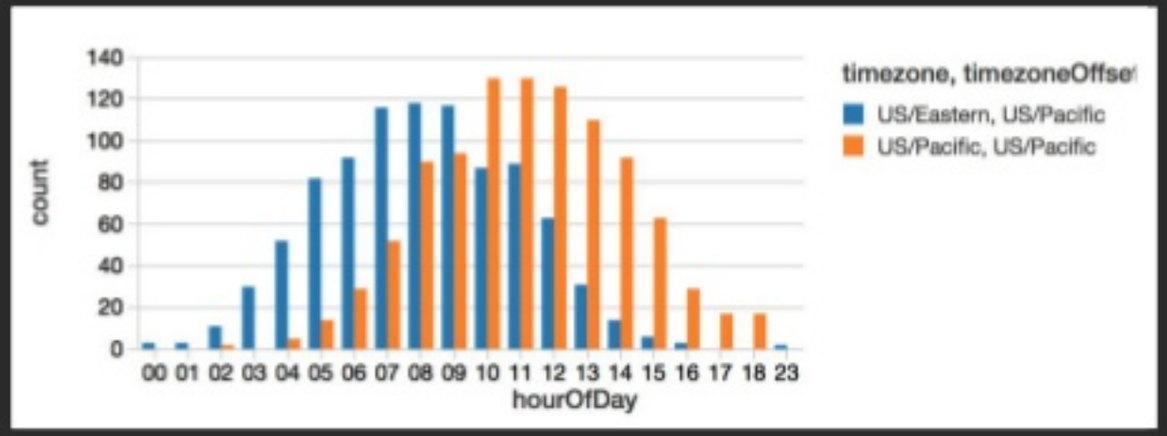
Extraction Code

```scala
package com.fis.mobile.examples

import ...

class ExtractionClassV1(sparkSession: SparkSession) {
  val TIMESTAMP_FORMAT = "yyyy-MM-dd HH:mm:ss"

  def RunExtractJob(sourceFilePath: String, destinationFilePath: String, timezoneOffset: String): Unit = {
    val sourceDataFrame: DataFrame = GetJsonDataFrame(sourceFilePath)
    val extractedDataFrame: DataFrame = ExtractDataFrame(sourceDataFrame, timezoneOffset)
    SaveJsonDataFrame(extractedDataFrame, destinationFilePath)
  }

  def GetExtractedJsonDataFrame(filePath: String, timezoneOffset: String): DataFrame = {
    ExtractDataFrame(GetJsonDataFrame(filePath), timezoneOffset)
  }

  private def GetJsonDataFrame(filePath: String): DataFrame = {
    sparkSession.read.json(filePath)
  }

  private def ExtractDataFrame(dataFrame: DataFrame, timezoneOffset: String): DataFrame = {
    import sparkSession.implicits._

    dataFrame
      .withColumn("timezoneOffset", lit(timezoneOffset))
      .withColumn("timestampGmt", from_unixtime($"unixTimestamp"))
      .withColumn("timestampLtz",
        date_format(from_utc_timestamp(from_unixtime($"unixTimestamp"), timezoneOffset), TIMESTAMP_FORMAT))
  }

  private def SaveJsonDataFrame(dataFrame: DataFrame, filePath: String): Unit = {
    dataFrame.write.json(filePath)
  }
}
```

SQL Validation Test Code

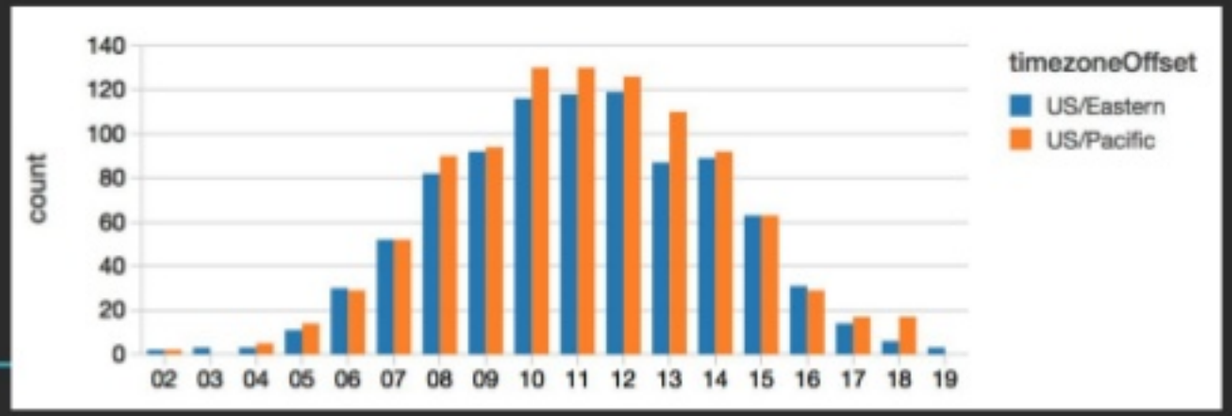
```
SELECT  TIMEZONE_OFFSET,
    TIMESTAMP_GMT,
    TIMESTAMP_LTZ
FROM
    APPSTORE_REVIEWS
WHERE
    TIMEZONE_OFFSET <> 0
LIMIT 10;
```

Cucumber Test




```
@Extraction @TempFileCleanup @ApacheSpark
Feature: Json Logs Extract Process V1

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

  @V1
  Scenario: Basic extraction of Epoch time into readable local time zones specified by person doing extraction
    Given there is a file "srcFolder/example.json" with the following lines:
      | {"logId":1, "unixTimestamp":1459482142, "timezone":"US/Pacific"} |
      | {"logId":2, "unixTimestamp":1459482142, "timezone":"US/Eastern"} |
    When the method RunExtractJobV1 gets called with
      | SourceFolder      | srcFolder/* |
      | DestinationFolder | dstFolder   |
      | TimezoneOffset    | US/Pacific  |
    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|timestampLtz       |timezoneOffset|
      |1    |1459482142   |2016-03-31 20:42:22|US/Pacific    |
      |2    |1459482142   |2016-03-31 20:42:22|US/Pacific    |

```

Cucumber Test Version 2




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

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

  Scenario: Basic extraction of Epoch time into readable local time zones of the devices
    Given there is a file "srcFolder2/example.json" with the following lines:
      | {"logId":1, "unixTimestamp":1459482142, "timezone":"US/Pacific"} |
      | {"logId":2, "unixTimestamp":1459482142, "timezone":"US/Eastern"} |
    When the method RunExtractJobV2 gets called with
      | SourceFolder      | srcFolder2/* |
      | DestinationFolder | dstFolder2   |
    Then there will be a "_SUCCESS" file in the "dstFolder2" folder
    And the folder "dstFolder2" will have json files with exactly the following DataFrame rows:
      |logId|unixTimestamp|timestampLtz       |timezoneOffset|
      |1    |1459482142   |2016-03-31 20:42:22|US/Pacific    |
      |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

```java
package com.fis.mobile.examples;

import ...

public class ExtractionStepDefinitions {
    @Given("^the system is in UTC time$")
    public void theSystemIsInGMTTime() throws Throwable {
        TimeZone.setDefault(TimeZone.getTimeZone("UTC"));
    }

    @Given("^there is a file \"([^\"]*)\" with the following lines:$")
    public void thereIsAFileWithTheFollowingLines(String propertiesPath, List<String> lines) throws Throwable {
        File file = new File(Helpers.getTempTestPath(propertiesPath));
        //noinspection ResultOfMethodCallIgnored
        file.getParentFile().mkdirs();
        PrintWriter writer = new PrintWriter(file.getAbsolutePath(), csn: "UTF-8");
        for (String str : lines) {
            writer.println(str.trim());
        }
        writer.close();
    }

    @When("^the method (RunExtractJobV1|RunExtractJobV2) gets called with$")
    public void theMethodRunExtractJobGetsCalledWith(String jobName, Map<String, String> arguments) throws Throwable {
        if(jobName.compareTo( anotherString: "RunExtractJobV1")==0) {
            new ExtractionClassV1(Helpers.testSparkSession).RunExtractJob(
                    Helpers.getTempTestPath(arguments.get("SourceFolder")),
                    Helpers.getTempTestPath(arguments.get("DestinationFolder")),
                    arguments.get("TimezoneOffset"));
        }
        else if(jobName.compareTo( anotherString: "RunExtractJobV2")==0) {
            new ExtractionClassV2(Helpers.testSparkSession).RunExtractJob(
                    Helpers.getTempTestPath(arguments.get("SourceFolder")),
                    Helpers.getTempTestPath(arguments.get("DestinationFolder")));
        } else { throw new PendingException(); }
    }
```

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

```java
package com.fis.mobile.examples;

import ...

@RunWith(Cucumber.class)
@CucumberOptions(
        glue = "com.fis.mobile.examples",
        features = {"src/test/resources/Feature/"}
)
public class RunFeatures {

}
```

```java
package com.fis.mobile.examples;

import ...

public class BeforeAfterHooks {
    @Before("@TempFileCleanup")
    @After("@TempFileCleanup")
    public void CleanupTempFiles() throws IOException, InterruptedException {
        File tempDirectory = new File(Helpers.getTempPath( relativePath: ""));
        FileUtils.deleteDirectory(tempDirectory);
    }

    @Before("@ApacheSpark")
    public void setup() {
        if(Helpers.testSparkSession == null) {
            Helpers.testSparkSession = SparkSession
                    .builder()
                    .master("local")
                    .appName("cucumberSparkAppTestSession")
                    .config("spark.driver.host", "127.0.0.1")
                    .getOrCreate();
        }
    }
}
```

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

2 Kinds of People in this world…

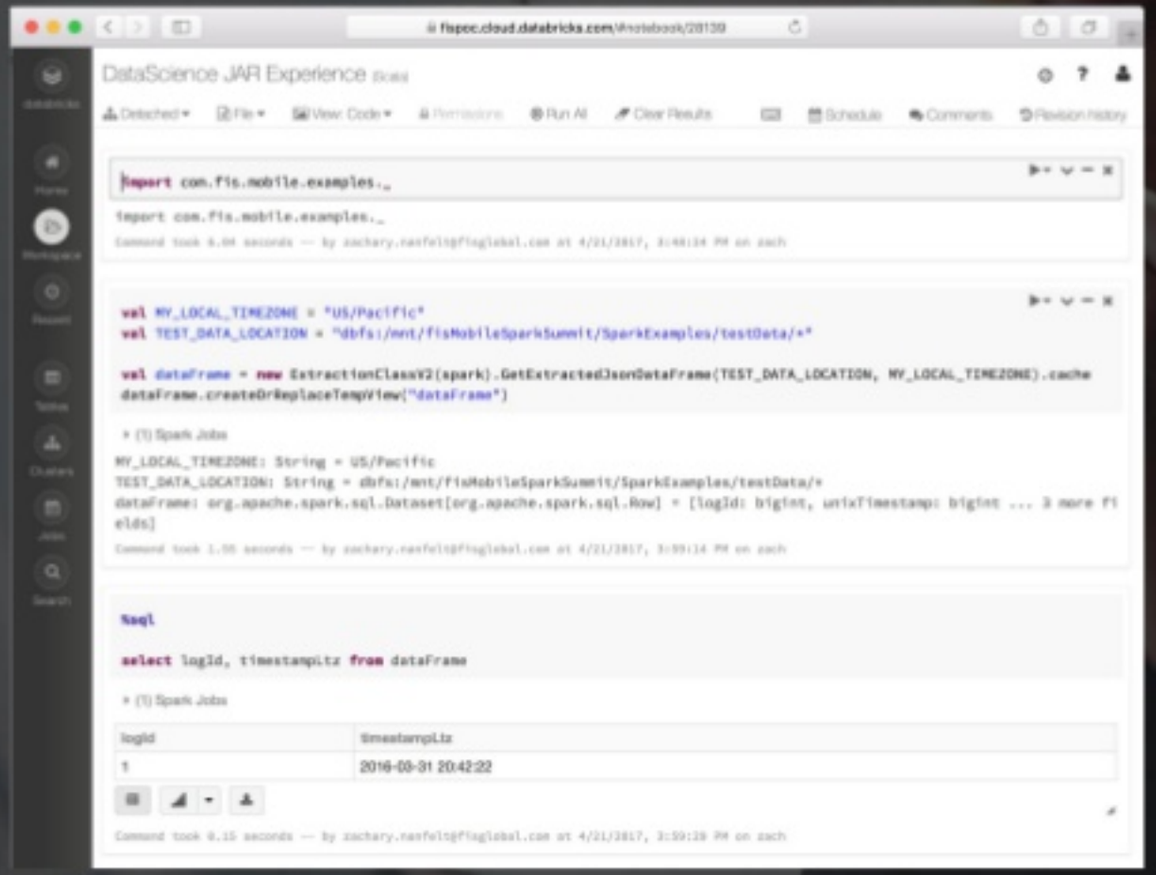
About cucumber

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.

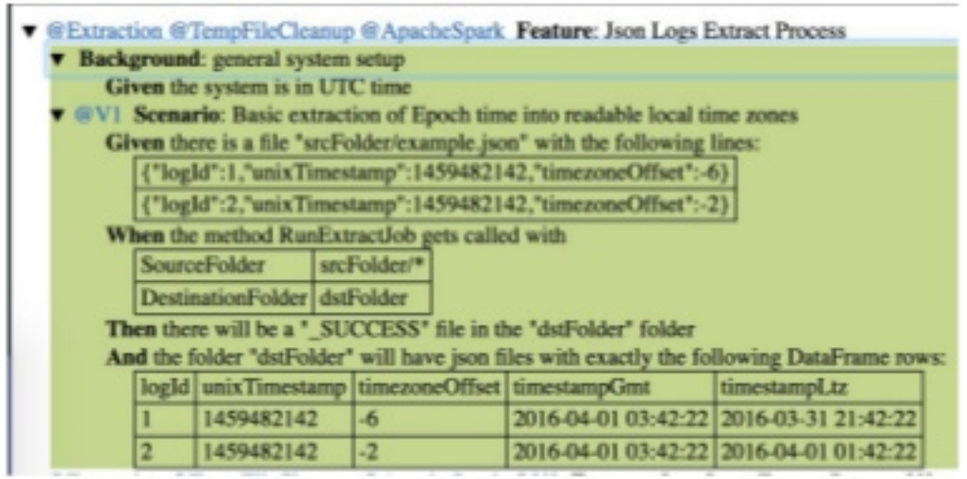
Code

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



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

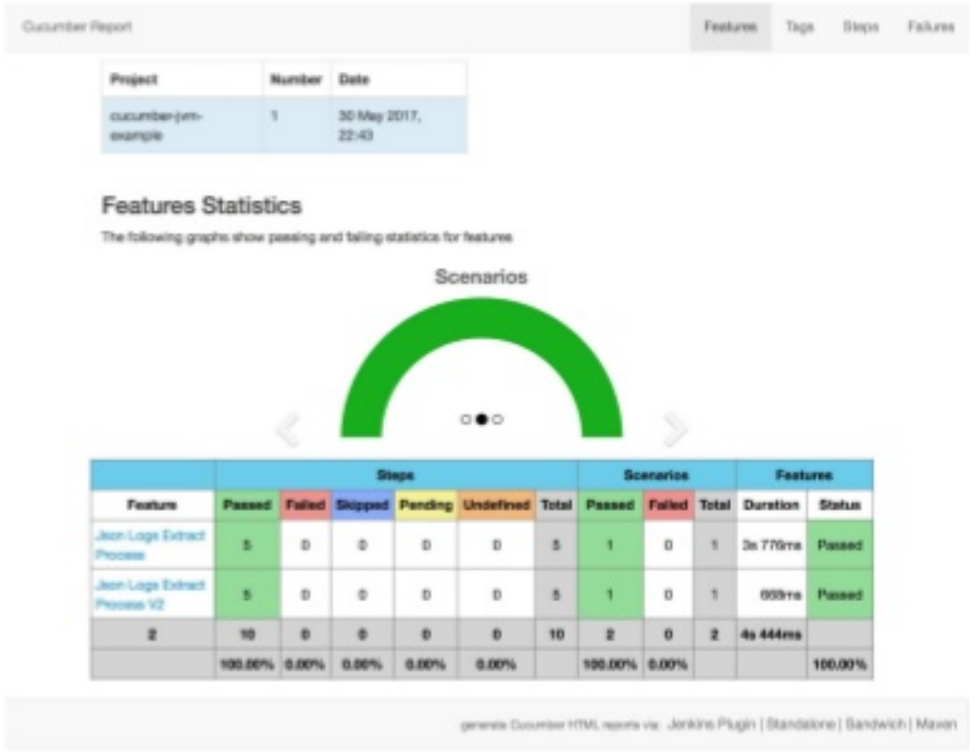
SourceFolder	srcFolder/*
DestinationFolder	dstFolder

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



Cucumber Report — Features | Tags | Steps | Failures

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

Features Statistics

The following graphs show passing and failing statistics for features

Scenarios

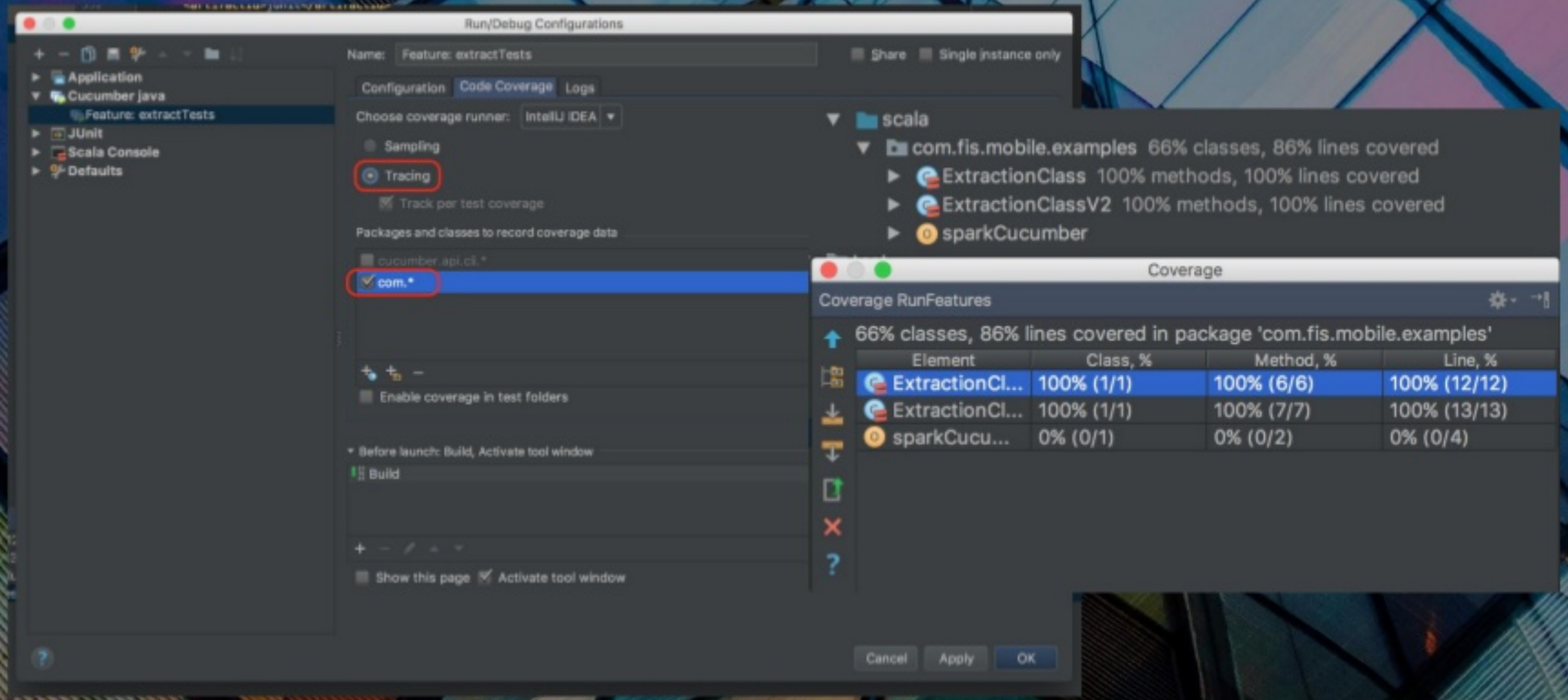
	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
Json Logs Extract Process	5	0	0	0	0	5	1	0	1	3s 770ms	Passed
Json Logs Extract Process V2	5	0	0	0	0	5	1	0	1	693ms	Passed
2	10	0	0	0	0	10	2	0	2	4s 444ms	
	100.00%	0.00%	0.00%	0.00%	0.00%		100.00%	0.00%			100.00%

generate Cucumber HTML reports via: Jenkins Plugin | Standalone | Sandwich | Maven

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)



Resources, Resources, Resources

- cucumber.io
- cucumber-reporting
- Pragmatic Books
  - Cucumber Book
  - Cucumber Recipes
  - Cucumber for Java
- Specification by Example
- Databricks Blog

Thank you