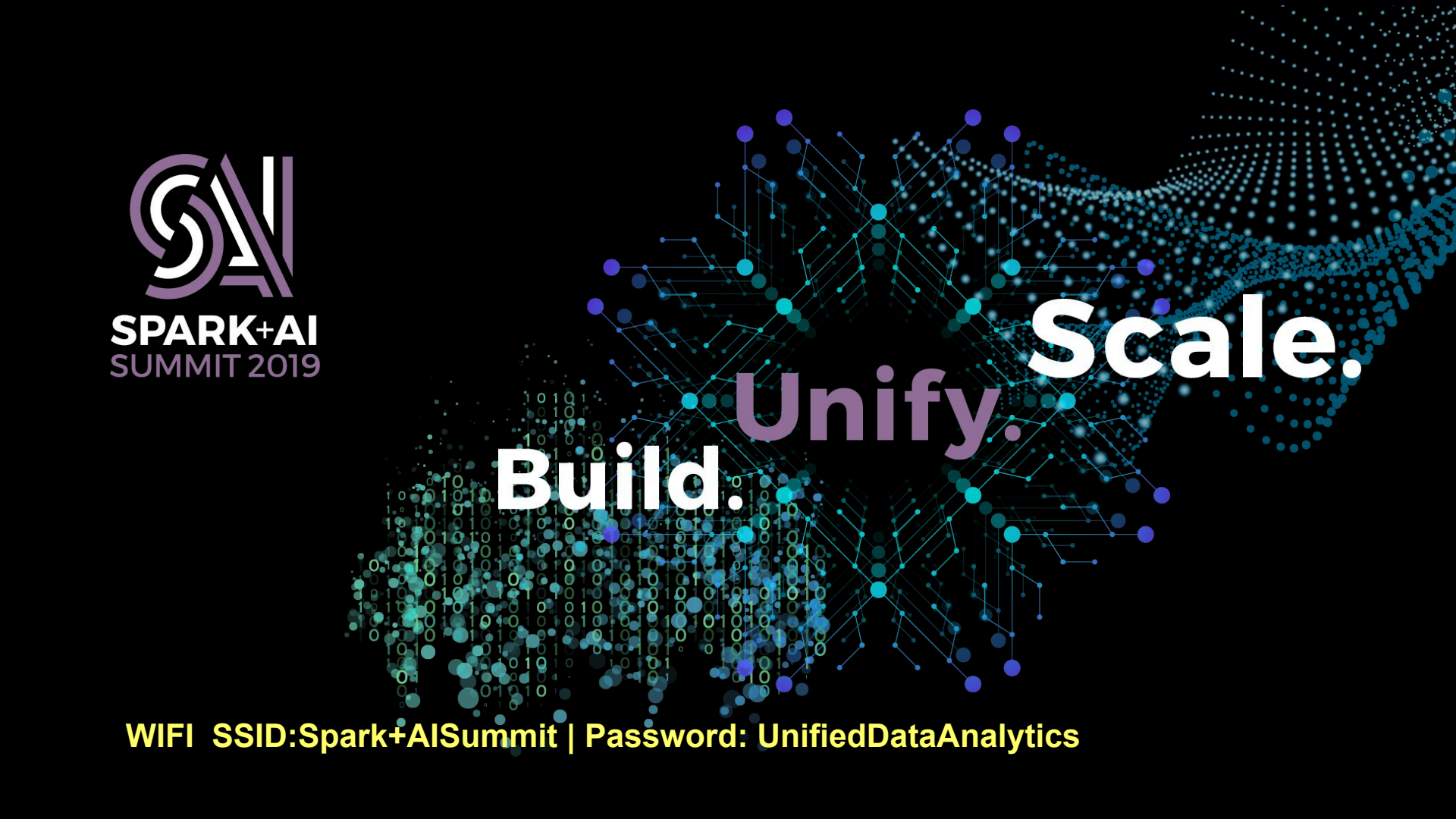
SPARK+AI
SUMMIT 2019
Build.
Unify.
Scale.
WIFI SSID:Spark+AISummit | Password: UnifiedDataAnalytics

From `HelloWorld` to Configurable and Reusable Apache Spark Applications In Scala

Oliver Ţupran

#UnifiedDataAnalytics #SparkAISummit

whoami

Oliver Ţupran

Software Engineer

Aviation, Banking, Telecom...

Scala Enthusiast

Apache Spark Enthusiast


olivertupran


tupol


@olivertupran

Intro

Audience

- Professionals starting with Scala and Apache Spark
- Basic Scala knowledge is required
- Basic Apache Spark knowledge is required

Agenda

- Hello, World!
- Problems
- Solutions
- Summary

Intro

Hello, World!

```
./bin/spark-shell

scala> val textFile = spark.read.textFile("README.md")

textFile: org.apache.spark.sql.Dataset[String] = [value: string]

scala> val linesWithSpark = textFile.filter(line => line.contains("Spark"))

linesWithSpark: org.apache.spark.sql.Dataset[String] = [value: string]

scala> linesWithSpark.count() // How many lines contain "Spark"?

res3: Long = 15
```

Source spark.apache.org/docs/latest/quick-start.html

Hello, World!

```scala
object SimpleApp {
  def main(args: Array[String]) {
    val spark = SparkSession.builder.appName("Simple Application").getOrCreate()
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
    val logData = spark.read.textFile(logFile).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println(s"Lines with a: $numAs, Lines with b: $numBs")
    spark.stop()
  }
}
```

Source spark.apache.org/docs/latest/quick-start.html

Problems

- Configuration mixed with the application logic
- IO can be much more complex than it looks
- Hard to test

Solutions

- Clean separation of the business logic
- Spark session out of the box
- Configuration and validation support
- Encourage and facilitate testing

[tupol/spark-utils]

Business Logic Separation

```scala
/**
 * @tparam Context The type of the application context class.
 * @tparam Result The output type of the run function.
 */
trait SparkRunnable[Context, Result] {
  /**
   * @param context context instance containing all the application specific configuration
   * @param spark active spark session
   * @return An instance of type Result
   */
  def run(implicit spark: SparkSession, context: Context): Result
}
```

Source github.com/tupol/spark-utils

Solutions

Stand-Alone App Blueprint

```
trait SparkApp[Context, Result] extends SparkRunnable[Context, Result] with Logging {
  def appName: String = . . .
  private def applicationConfiguration(implicit spark: SparkSession, args: Array[String]):
     com.typesafe.config.Config = . . .
  def createSparkSession(runnerName: String): SparkSession =
    . . .
  def createContext(config: com.typesafe.config.Config): Context
  def main(implicit args: Array[String]): Unit = {
     // Create a SparkSession, initialize a Typesafe Config instance,
     // validate and initialize the application context,
     // execute the run() function, close the SparkSession and
     // return the result or throw and Exception
    . . .
  }
}
```

Source github.com/tupol/spark-utils

Solutions

Back to SimpleApp

```scala
object SimpleApp {
  def main(args: Array[String]) {
    val spark = SparkSession.builder.appName("Simple Application").getOrCreate()
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
    val logData = spark.read.textFile(logFile).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println(s"Lines with a: $numAs, Lines with b: $numBs")
    spark.stop()
  }
}
```

Source spark.apache.org/docs/latest/quick-start.html

SimpleApp as SparkApp


```scala
object SimpleApp extends SparkApp[Unit, Unit]{
  override def createContext(config: Config): Unit = Unit
  override def run(implicit spark: SparkSession, context: Unit): Unit {
    val logFile = "YOUR_SPARK_HOME/README.md"
    val logData = spark.read.textFile(logFile).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println(s"Lines with a: $numAs, Lines with b: $numBs")
  }
}
```

Source github.com/tupol/spark-utils-demos/

Solutions

SimpleApp as SparkApp


```scala
object SimpleApp extends SparkApp[Unit, Unit]{
  override def createContext(config: Config): Unit = Unit
  override def run(implicit spark: SparkSession, context: Unit): Unit = {
    val logFile = "YOUR_SPARK_HOME/README.md"  // Should be some file on your system
    val logData = spark.read.textFile(logFile).cache()
    val (numAs, numBs) = appLogic(logData)
    println(s"Lines with a: $numAs, Lines with b: $numBs")
  }
  def appLogic(data: Dataset[String]): (Long, Long) = {
    val numAs = data.filter(line => line.contains("a")).count()
    val numBs = data.filter(line => line.contains("b")).count()
    (numAs, numBs)
  }
}
```

Source github.com/tupol/spark-utils-demos/

Solutions

SimpleApp as SparkApp


```
case class SimpleAppContext(input: FileSourceConfiguration , filterA: String, filterB: String)

object SimpleApp extends SparkApp[SimpleAppContext, Unit]{
  override def createContext(config: Config): SimpleAppContext = ???
  override def run(implicit spark: SparkSession, context: SimpleAppContext): Unit = {
    val logData = spark.source(context.input).read.as[String].cache
    val (numAs, numBs) = appLogic(logData)
    println(s"Lines with a: $numAs, Lines with b: $numBs")
  }
  def appLogic(data: Dataset[String], context: SimpleAppContext): (Long, Long) = {
    val numAs = data.filter(line => line.contains(context.filterA)).count()
    val numBs = data.filter(line => line.contains(context.filterB)).count()
    (numAs, numBs)
  }
```

Source github.com/tupol/spark-utils-demos/

Solutions

Why Typesafe Config?

- supports files in three formats: Java properties, JSON, and a **human-friendly JSON superset**
- merges multiple files across all formats
- can load from files, URLs or classpath
- users can override the config with Java system properties, `java -Dmyapp.foo.bar=10`
- supports configuring an app, with its framework and libraries, all from a single file such as `application.conf`
- **extracts typed properties**
- JSON superset features:
  - comments
  - includes
  - **substitutions (`"foo" : ${bar}, "foo" : Hello ${who}`)**
  - properties-like notation (`a.b=c`)
  - **less noisy, more lenient syntax**
  - substitute environment variables (`logdir=${HOME}/logs`)
  - **lists**

Source github.com/lightbend/config

Solutions

Application Configuration File

Hocon

```
SimpleApp {
  input {
    format: text
    path: SPARK_HOME/README.md
  }
  filterA: A
  filterB: B
}
```

Java Properties

```
SimpleApp.input.format=text
SimpleApp.input.path=SPARK_HOME/README.md
SimpleApp.filterA=A
SimpleApp.filterB=B
```

Solutions

Configuration and Validation

```
case class SimpleAppContext(input: FileSourceConfiguration , filterA: String, filterB: String)

object SimpleAppContext extends Configurator[SimpleAppContext] {
  import org.tupol.utils.config._
  override def validationNel(config: com.typesafe.config.Config):
                    scalaz .ValidationNel [Throwable, SimpleAppContext ] = {
    config.extract[FileSourceConfiguration]("input")
      .ensure(new IllegalArgumentException (
        "Only 'text' format files are supported" ).toNel)(_.format == FormatType.Text) |@|
    config.extract[ String]("filterA") |@|
    config.extract[ String]("filterB") apply
    SimpleAppContext .apply
  }
}
```

Source github.com/tupol/spark-utils-demos/

Solutions

`Configurator` Framework?

- DSL for easy definition of the context
  - **`config.extract[Double]("parameter.path")`**
  - `|@|` operator to compose the extracted parameters
  - **`apply`** to build the configuration case class
- Type based configuration parameters extraction
  - **`extract[Double]("parameter.path")`**
  - **`extract[Option[Seq[Double]]]("parameter.path")`**
  - **`extract[Map[Int, String]]("parameter.path")`**
  - **`extract[Either[Int, String]]("parameter.path")`**
- Implicit **`Configurators`** can be used as extractors in the DSL
  - **`config.extract[SimpleAppContext]("configuration.path")`**
- The `ValidationNel` contains either a list of exceptions or the application context

Solutions

SimpleApp as SparkApp


```
case class SimpleAppContext(input: FileSourceConfiguration, filterA: String, filterB: String)

object SimpleApp extends SparkApp[SimpleAppContext, (Long, Long)]{
  override def createContext(config: Config): SimpleAppContext = SimpleAppContext(config).get
  override def run(implicit spark: SparkSession, context: SimpleAppContext): (Long, Long) = {
    val logData = spark.source(context.input).read.as[ String].cache
    val (numAs, numBs) = appLogic(logData)
    println(s"Lines with a: $numAs, Lines with b: $numBs")
    (numAs, numBs)
  }
  def appLogic(data: Dataset[String], context: SimpleAppContext): (Long, Long) = {
    . . .
  }
}
```

Source github.com/tupol/spark-utils-demos/

Solutions

Data Sources and Data Sinks

CSV	JSON	XML	JDBC	...
format URI (connection URL, file path…) *schema*				
sep encoding quote escape comment header inferSchema ignoreLeadingWhiteSpace ignoreTrailingWhiteSpace nullValue nanValue positiveInf negativeInf dateFormat timestampFormat maxColumns maxCharsPerColumn maxMalformedLogPerPartition mode	primitivesAsString prefersDecimal allowComments allowUnquotedFieldNames allowSingleQuotes allowNumericLeadingZeros allowBackslashEscapingAnyCharacter mode columnNameOfCorruptRecord dateFormat timestampFormat	rowTag samplingRatio excludeAttribute treatEmptyValuesAsNulls mode columnNameOfCorruptRecord attributePrefix valueTag charset ignoreSurroundingSpaces	table columnName lowerBound upperBound numPartitions connectionProperties	

Source spark.apache.org/docs/latest/

Data Sources and Data Sinks

```
import org.tupol.spark.implicits._
import org.tupol.spark.io._
import spark.implicits._
. . .
val input = config.extract[FileSourceConfiguration]("input").get
val lines = spark.source(input).read.as[String]
// org.tupol.spark.io.FileDataSource(input).read
// spark.read.format(...).option(...).option(...).schema(...).load()

val output = config.extract[FileSinkConfiguration]("output").get
lines.sink(output).write
// org.tupol.spark.io.FileDataSink(output).write(lines)
// lines.write.format(...).option(...).option(...).partitionBy(...).mode(...)
```

Solutions

Data Sources and Data Sinks

- Very concise and intuitive DSL
- Support for multiple formats: text, csv, json, xml, avro, parquet, orc, jdbc, ...
- Specify a schema on read
- Schema is passed as a full json structure, as serialised by the **`StructType`**
- Specify the partitioning and bucketing for writing the data
- Structured streaming support
- Delta Lake support
- . . .

Solutions

Test! Test! Test!

```
class SimpleAppSpec extends FunSuite with Matchers with SharedSparkSession  {
 . . .
 val DummyInput = FileSourceConfiguration("no path", TextSourceConfiguration())
 val DummyContext = SimpleAppContext(input = DummyInput, filterA = "", filterB = "")

 test("appLogic should return 0 counts of a and b for an empty DataFrame") {
   val testData = spark.emptyDataset[String]
   val result = SimpleApp.appLogic(testData, DummyContext)
   result shouldBe (0, 0)
 }
 . . .
}
```

Source github.com/tupol/spark-utils-demos/

Test! Test! Test!

```
class SimpleAppSpec extends FunSuite with Matchers with SharedSparkSession  {
 . . .

 test("run should return (1, 2) as count of a and b for the given data") {
   val inputSource = FileSourceConfiguration("src/test/resources/input-test-01",
                                             TextSourceConfiguration())
   val context = SimpleAppContext(input = inputSource, filterA = "a", filterB = "b")
   val result = SimpleApp.run(spark, context)
   result shouldBe (1, 2)
 }
 . . .
}
```

Source github.com/tupol/spark-utils-demos/

Solutions

Format Converter

```
case class MyAppContext(input : FormatAwareDataSourceConfiguration,
                        output: FormatAwareDataSinkConfiguration)

object MyAppContext extends Configurator[MyAppContext] {
 import scalaz.ValidationNel
 import scalaz.syntax.applicative._
 def validationNel(config: Config): ValidationNel[Throwable, MyAppContext] = {
   config.extract[FormatAwareDataSourceConfiguration]("input") |@|
     config.extract[FormatAwareDataSinkConfiguration]("output") apply
     MyAppContext.apply
 }
}
```

Source github.com/tupol/spark-utils-demos/

Solutions

Format Converter

```
object MyApp extends SparkApp[MyAppContext, DataFrame] {

 override def createContext(config: Config): MyAppContext = MyAppContext(config).get

 override def run(implicit spark: SparkSession, context: MyAppContext): DataFrame = {
   val data = spark.source(context.input).read
   data.sink(context.output).write
 }
}
```

Source github.com/tupol/spark-utils-demos/

Solutions

Beyond Format Converter

```
object MyApp extends SparkApp[MyAppContext, DataFrame] {

 override def createContext(config: Config): MyAppContext = MyAppContext(config).get

 override def run(implicit spark: SparkSession, context: MyAppContext): DataFrame = {
   val inputData = spark.source(context.input).read
   val outputData = transform(inputData)
   outputData.sink(context.output).write
 }
 def transform(data: DataFrame)(implicit spark: SparkSession, context: MyAppContext) = {
   data  // Transformation logic here
 }
}
```

Source github.com/tupol/spark-utils-demos/

Solutions

Summary

- Write Apache Spark applications with minimal ceremony
  - batch
  - structured streaming
- IO and general application configuration support
- Facilitates testing
- Increase productivity

tupol/spark-utils spark-tools spark-utils-demos spark-apps.seed.g8

What's Next?

- Support for more source types
- Improvements of the configuration framework
- Feedback is welcomed!
- Help is welcomed!


olivertupran


tupol


@olivertupran

References

Presentation	https://tinyurl.com/yxuneqcs
spark-utils	https://github.com/tupol/spark-utils
spark-utils-demos	https://github.com/tupol/spark-utils-demos
spark-apps.seed.g8	https://github.com/tupol/spark-apps.seed.g8
spark-tools	https://github.com/tupol/spark-tools
Lightbend Config	https://github.com/lightbend/config
Giter8	http://www.foundweekends.org/giter8/
Apache Spark	http://spark.apache.org/
ScalaZ	https://github.com/scalaz/scalaz
Scala	https://scala-lang.org

SPARK+AI SUMMIT 2019

DON'T FORGET TO RATE AND REVIEW THE SESSIONS

SEARCH SPARK + AI SUMMIT

Download on the App Store

GET IT ON Google Play