inovex

The productivity booster for your Data Projects with Scala 3 & scala-cli

こんにちは - About me

Kolja Maier

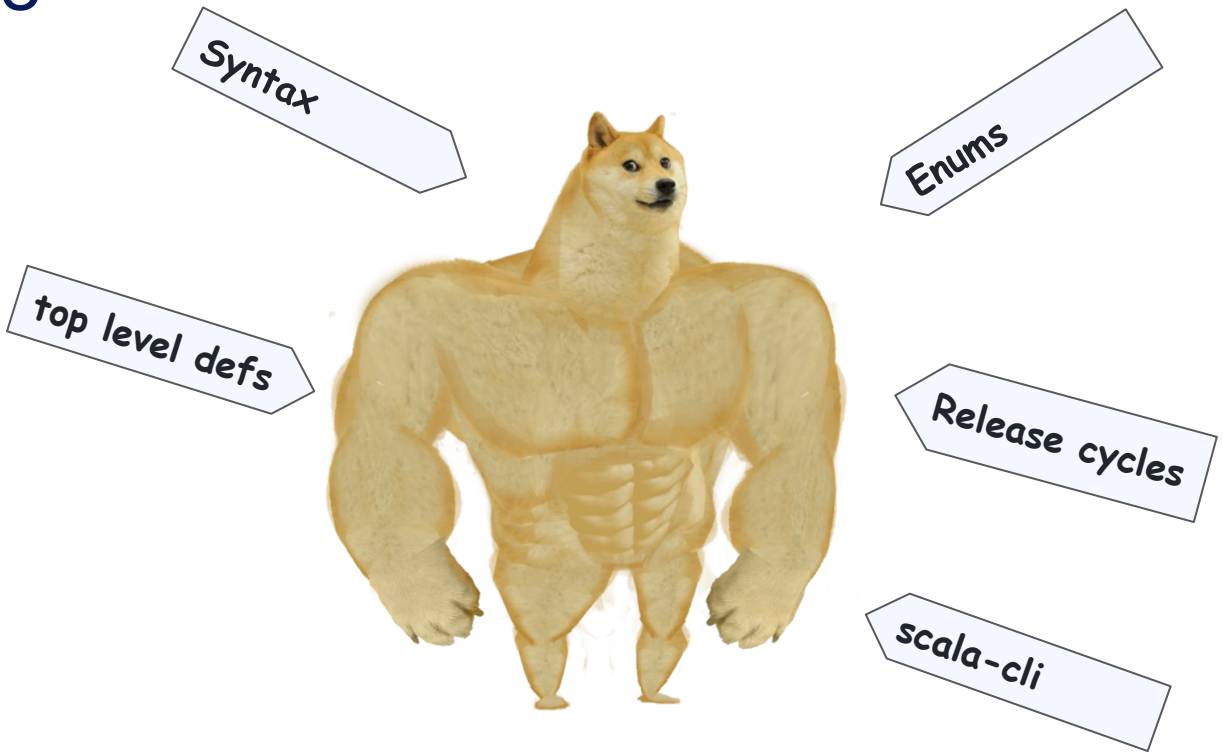
- Data & ML Engineering @ inovex 🇩🇪
- ~10 years of Scala experience

Big Data

Battle proof architecture

Productivity

Scala 3



inovex

scala-cli examples

Spark

```
scala-cli run --spark SparkExample.scala
```

```
people
  .toDS
  .group
        groupBy(cols: Column*): RelationalGroup…
        groupBy(col1: String, cols: String*): R…
        groupByKey[K](func: MapFunction[Person,…
        groupByKey[K: Encoder](func: Person => …
```

```
(cols: Column*): RelationalGrou
pedDataset

Groups the Dataset using the specified
columns, so we can run aggregation on
them. See RelationalGroupedDataset for all
the available aggregate functions.

// Compute the average for all num
ds.groupBy($"department").avg()

// Compute the max age and average
ds.groupBy($"department", $"gender
  "salary" -> "avg",
  "age" -> "max"
))
```

Spark

```
people.toDS
  .groupByKey(_.name)
  .mapGroups { (name, peopleWithSameName) =>
```

```
+--------+---+---+----+
|      _1| _2| _3|  _4|
+--------+---+---+----+
|Marukami| 49| 75|63.0|
| Mishima| 25| 25|25.0|
|  Suzuki| 16| 41|28.5|
+--------+---+---+----+
```

Spark

```scala
people.toDS
  .groupByKey(_.name)
  .mapGroups { (name, peopleWithSameName) =>
    val ages = peopleWithSameName.map(_.age).toList
    (
      name,
      ages.min,
      ages.max,
      ages.sum.toDouble / ages.length
    )
  }
```

```
+--------+---+---+----+
|      _1| _2| _3|  _4|
+--------+---+---+----+
|Marukami| 49| 75|63.0|
| Mishima| 25| 25|25.0|
|  Suzuki| 16| 41|28.5|
+--------+---+---+----+
```

Spark

```scala
people.toDS
  .groupByKey(_.name)
  .mapGroups { (name, peopleWithSameName) =>
    val ages = peopleWithSameName.map(_.age).toList
    (
      name,
      ages.min,
      ages.max,
      ages.sum.toDouble / ages.length
    )
  }
  .map(statistics =>
    (statistics._1, statistics._2, statistics._3, statistics._4)
  )
  .show()
```

```
+--------+---+---+----+
|      _1| _2| _3|  _4|
+--------+---+---+----+
|Marukami| 49| 75|63.0|
| Mishima| 25| 25|25.0|
|  Suzuki| 16| 41|28.5|
+--------+---+---+----+
```

Spark

```
people.toDS.createOrReplaceTempView("people")

val sqlDF =
  spark.sql(
    """
    SELECT name, COUNT(*) AS cnt
    FROM people
    GROUP BY name
    """
  )
sqlDF.show()
```

```
+--------+---+
|    name|cnt|
+--------+---+
|  Suzuki|  2|
| Mishima|  1|
|Marukami|  3|
+--------+---+
```

Spark

```
//> using scala 3
//> using dep org.apache.spark:spark-sql-api_2.13:3.5.1
//> using dep io.github.vincenzobaz::spark-scala3-encoders:0.2.6

import org.apache.spark.sql.SparkSession
import scala3encoders.given
```

BigQuery UDFs

```
//> using scala 3
//> using platform js

```

BigQuery UDFs

```
//> using scala 3
//> using platform js

import scala.scalajs.js
import scala.scalajs.js.annotation.*

```

BigQuery UDFs

```
//> using scala 3
//> using platform js

import scala.scalajs.js
import scala.scalajs.js.annotation.*

enum Flavor:
  case 🥬, 🐓, 🐄, 🐏, 🐖
```

BigQuery UDFs

```scala
//> using scala 3
//> using platform js

import scala.scalajs.js
import scala.scalajs.js.annotation.*

enum Flavor:
  case 🥬, 🐓, 🐄, 🦪, 🐖

@JSExportTopLevel("dailyFlavor")
def dailyFlavor(b: String): String =
  Flavor.valueOf(b) match
    case Flavor.`🥬` => "Veggie day"
    case Flavor.`🦪` => "Seafood lover!"
    case _ => "...and so on"
```

BigQuery UDFs

```
CREATE TEMP FUNCTION dailyFlavor(f STRING)
RETURNS STRING
```

BigQuery UDFs

```
CREATE TEMP FUNCTION dailyFlavor(f STRING)
RETURNS STRING
LANGUAGE js
OPTIONS (
  library=['gs://$GCS_BUCKET/main.js'])
```

BigQuery UDFs

```
CREATE TEMP FUNCTION dailyFlavor(f STRING)
RETURNS STRING
LANGUAGE js
OPTIONS (
  library=['gs://$GCS_BUCKET/main.js'])
  AS r"""
    return dailyFlavor(f)
  """;
```

BigQuery UDFs

```
CREATE TEMP FUNCTION dailyFlavor(f STRING)
RETURNS STRING
LANGUAGE js
OPTIONS (
  library=['gs://$GCS_BUCKET/main.js'])
  AS r"""
    return dailyFlavor(f)
  """;
  SELECT dailyFlavor('🥬') AS result;
```

result
Veggie day

Kafka - Data modelling

```
1 case class Ramen(brand: String, flavor: Flavor, price: Double)
```

```
1 enum Flavor:
2   case 🥬, 🐓, 🐄, 🐟, 🐖
```

Kafka - Data modelling

```scala
case class Ramen(brand: String, flavor: Flavor, price: Double)
object Ramen:
  given JsonCodec[Ramen] = DeriveJsonCodec.gen[Ramen]
```

```scala
enum Flavor:
  case 🥬, 🐓, 🐄, 🦐, 🐖
object Flavor:
  given JsonCodec[Flavor] =
    JsonCodec[String].transform(Flavor.valueOf, _.toString)
```

Kafka - Data modelling

```scala
case class Ramen(brand: String, flavor: Flavor, price: Double)
object Ramen:
  given JsonCodec[Ramen] = DeriveJsonCodec.gen[Ramen]
  val valueSerializer = Serde.string.inmapM[Any, Ramen](s =>
    ZIO
      .fromEither(s.fromJson[Ramen])
      .mapError(e => new RuntimeException(e))
  )(r => ZIO.succeed(r.toJson))
```

```scala
enum Flavor:
  case 🥬, 🐓, 🐄, 🦐, 🐖
object Flavor:
  given JsonCodec[Flavor] =
    JsonCodec[String].transform(Flavor.valueOf, _.toString)
```

Kafka - Producer

```
ZStream
  .fromZIO(Random.nextIntBetween(0, Int.MaxValue))
  .forever
```

Kafka - Producer

```
ZStream
  .fromZIO(Random.nextIntBetween(0, Int.MaxValue))
  .forever
  .mapZIO { id =>
    Producer.produce[Any, Int, Ramen](
      topic = "ramen",
      key = id,
```

Kafka - Producer

```
ZStream
  .fromZIO(Random.nextIntBetween(0, Int.MaxValue))
  .forever
  .mapZIO { id =>
    Producer.produce[Any, Int, Ramen](
      topic = "ramen",
      key = id,
      value = Ramen(
        SRandom.shuffle(List("Nissin", "Ichiran", "Samyang")).head,
        Flavor.fromOrdinal(id % 5),
        SRandom.between(0.0, 1.0)
      ),
```

Kafka - Producer

```
ZStream
  .fromZIO(Random.nextIntBetween(0, Int.MaxValue))
  .forever
  .mapZIO { id =>
    Producer.produce[Any, Int, Ramen](
      topic = "ramen",
      key = id,
      value = Ramen(
        SRandom.shuffle(List("Nissin", "Ichiran", "Samyang")).head,
        Flavor.fromOrdinal(id % 5),
        SRandom.between(0.0, 1.0)
      ),
      keySerializer = Serde.int,
      Ramen.valueSerializer
    )
  }
```

```
Received Ramen(Ichiran,🐓,0.022659367901554872)
Sent value ramen-0@32776
Received Ramen(Ichiran,🥬,0.2837352181068149)
Sent value ramen-0@32777
^CSent value ramen-0@32778
Sent value ramen-0@32779
Sent value ramen-0@32780
Received Ramen(Samyang,🐖,0.15919173807975895)
Sent value ramen-0@32781
Sent value ramen-0@32782
Received Ramen(Ichiran,🐖,0.3246139886443965)
Sent value ramen-0@32783
Received Ramen(Nissin,🐓,0.14962751137279784)
Sent value ramen-0@32784
Received Ramen(Samyang,🦐,0.09595954071228774)
```

	Offset	Partition
−	0	0

Key | Value | Headers


```
{
  "brand": "Samyang",
  "flavor": "🐓",
  "price": 0.8224032060
}
```

Summary

Summary

- It is crucial to have a setup for validating data tooling with fast evolving business requirements
  - Scala 3 powerful features and scala-cli represent a strong combination!
  - Disclaimer: Use what fits best
- Further productivity pointers
  - scala-cli toolkit
  - just

Thank you!

Kolja Maier

Code to this talk

inovex