kogito-4-spark

Kogito run over rows, scaled by Spark

Chris Twiner

Copyright @ 2025

Table of contents

1. kogito-4-spark - 0.0.1-RC16-SNAPSHOT 3
1.1 Supported Runtimes 3
1.2 How to Use 3
1.3 Supported DMNContextProviders 4
1.4 Supported DMNResultProviders 4
2. Getting Started 7
2.1 Running kogito-4-spark on Databricks 7

Copyright @ 2025

1. kogito-4-spark - 0.0.1-RC16-SNAPSHOT

verage

Statement	74.24 %	Branch	48.93 %

A Kogito implementation of the dmn-4-spark API.

1.1 Supported Runtimes

Spark 3.5.x (Spark 4 hopefully coming soon) based runtimes on jdk 17 (OSS 2.13 builds are also provided).

Databricks requires the use of JNAME, with its associated reduction in support, in order to run on a non-jdk 8 VM for DBRs 14.0, 14.3 and 15.4. 16.4 moves to JDK 17 by default and also supports scala 2.13.

1.2 How to Use

Follow these instructions found on the API page to depend on the correct version.

Then, assuming your DMN files are available on the classpath (e.g. test/resources) define your files as:

```
val dmnFiles = Seq(
  DMNFile("common.dmn",
    this.getClass.getClassLoader.getResourceAsStream("common.dmn").readAllBytes()
  ),
  DMNFile("decisions.dmn",
    this.getClass.getClassLoader.getResourceAsStream("decisions.dmn").readAllBytes()
  )
)
```

or source them from byte arrays in a dataset using the api serialization.readVersionedFilesFromDF function.

Define the model you wish to run (the namespace and name must be present in one of the DMNFiles):

```
val dmnModel = DMNModelService(name, namespace, Some("DQService"), "struct<evaluate: array<boolean>>")
```

when the DecisionService is provided, DQService the above example, only it will be executed, using None will trigger evaluateAll semantics. The struct definition at the end defines your output structure, use JSON to serialize the DMNResult into JSON. You can also use the serialization.readVersionedModelServicesFromDF to load models.

Define the DMNInputfields:

```
val inputFields = Seq(
  DMNInputField("location", "String", "testData.location"),
  DMNInputField("idPrefix", "String", "testData.idPrefix"),
  DMNInputField("id", "Int", "testData.id"),
  DMNInputField("page", "Long", "testData.page"),
  DMNInputField("department", "String", "testData.department")
)
```

this use the fields location, idPrefix, id, page and department to set entries in the "testData" DMNContext map. Input fields can also be loaded via the serialization.readVersionedProvidersFromDF function.

Then combine the variables into the DMNExecution you wish to run with any additional configuration (currently ignored by kogito-4-spark):

```
val exec = DMNExecution(dmnFiles, service, inputFields, DMNConfiguration.empty /* default value */)
```

DMNExecutions too can be loaded from serialization.readVersionedExecutionsFromDF but requires you to provide all of the other input datasets (configuration can be optionally provided via serialization.readVersionedConfigurationDF).

Copyright @ 2025

finally register the DMN on your dataset (which contains the input fields):

```
val res = ds.withColumn("dmn", com.sparkutils.dmn.DMN.dmnEval(exec))
```

you may use the optional debug parameter to capture additional information from kogito's DMNResult.

NOTE: As with all Spark Datasets evaluation is lazy, write the dataset out to evaluate only once in-line per row as they are written.

1.3 Supported DMNContextProviders

The following JSON and DDL types are supported and provided to the org.kie.dmn.api.core.DMNContext

- JSON - string json representation
- String
- Integer
- Long
- Boolean
- Double
- Float
- Binary - provided as a byte[]
- Byte
- Short
- Date - provided as a LocalDate
- Timestamp - provided as a LocalDateTime
- Decimal - provided as DecimalType(DecimalType.MAX_PRECISION, DecimalType.DEFAULT_SCALE)
- struct<*> - with any nested types, provided as util.Map[String, Object] with field names as the keys
- array<*> - of any type, provided as util.List[Object]
- map - only supports util.Map[String, Object], the values may have any type

Non DDL Unary DMNContextProviders may be provided via a fully qualified class name and must provide a two arg constructor of DMNContextPath, Expression.

The data map used in the compilation of Context Providers can be configured via "useTreeMap=true" (default is false), this isn't terribly important for processing within Kogito but will affect JSON output ordering. (Interpreted mode is always ordered).

1.4 Supported DMNResultProviders

- JSON - Serializes the org.kie.dmn.api.core.DMNResult.getDecisionResults
- Struct<...> DDL - with each field representing a decision name to result mapping

Other DMNResultProviders may be provided via a fully qualified class name.

When Struct DDL is used each decisionName in the Kogito DMNResult will be stored against that struct, e.g. for a decision name "evaluate" which returns a list of booleans the DDL:

```
struct<evaluate: array<boolean>>
```

should be used. Where the decisionName is not present in the results null is used, each element will therefore be set to nullable by the library. Where a decision result is provided which is not the in the DDL it will be ignored (debug information may however be provided).

Use JSON to handle result schema evolution until a possible solution via Variants in Spark 4 is investigated.

Copyright @ 2025

1.4.1 Result Processing

In order to identify if a null result is due to an error or not a "_dmnEvalStatus: Byte" field can be added to the Struct DDL, e.g.:

```
struct<evaluate: array<boolean>, evaluate_dmnEvalStatus: Byte>
```

will store the Kogito DMNDecisionResult.getEvaluationStatus as a Byte with the following values:

DecisionEvaluationStatus (Severity)	**_dmnEvalStatus Int stored**
NOT_FOUND (kogito-4-spark only[1])	-6 (Typically a sign of a name mismatch)
NOT_EVALUATED	-5 (Should not happen)
EVALUATING	-4 (Should not happen)
SUCCEEDED	1
SKIPPED (WARN Msg.MISSING_EXPRESSION_FOR_DECISION)	-3
SKIPPED (ERROR)	-2
FAILED	0

These status' only replicate the Kogito DecisionEvaluationStatus usage and do not represent any business logic from the underlying DMN, that must of course be encoded in the result DLL directly.

The top level decision result map is proxied for both DDL and JSON processing, should this lead to a performance deficit you may disable it via the config option "fullProxyDS=false".

1.4.2 Debug mode

Use debugMode when calling evaluate to force the full DMNResult structure (without results) to be written out into an additional debugMode field, in the case where no issues are present this is likely overkill and should be kept for debug information only.
The debugMode field has the following DDL type (also found in ResultProcessors.debugDDL):

```
debugMode: array< struct<
  decisionId: String,
  decisionName: String,
  hasErrors: Boolean,
  messages: array< struct<
    sourceId: String,
    sourceReference: String,
    exception: String,
    feelEvent: struct<
      severity: String,
      message: String,
      line: Integer,
      column: Integer,
      sourceException: String,
      offendingSymbol: String
    >
  > >,
  evaluationStatus: String
> >
```

In this mode the output DDL more closely mimics the Kogito DMNResult, the two output types are not compatible.

The JSON provider when in debug mode serializes the entire DMNResult structure, when not the structure mimics the output of the Struct ddl counterpart e.g.:

```
{"eval":{"top1":"0a","strings":["a0i","b0i","c0i","d0i"],"structs":[{"a":"0","b":2061584302.16,"d":{"a":true,"b":true},"c":{"a1":"b1"}}]}}
```

becomes:

```
[{"decisionId":"_5BD6B443-5DB7-4CA4-84E2-AC86D643FB15","decisionName":"eval","result":{"top1":"0a","strings":["a0i","b0i","c0i","d0i"],"structs":
[{"a":"0","b":2061584302.16,"d":{"a":true,"b":true},"c":{"a1":"b1"}}]},"messages":[],"evaluationStatus":"SUCCEEDED"}]
```


Copyright @ 2025

1. The NOT_FOUND status is added by the library for the case where a _dmnEvalStatus field is provided in the ddl but this decision name that does not exist in the dmn. ↩

Last update: May 9, 2025 08:34:16

Created: May 9, 2025 08:34:16

Copyright @ 2025

2. Getting Started

2.1 Running kogito-4-spark on Databricks

The aim is to have explicit support for LTS', other interim versions may be supported as needed.

2.1.1 Running on Databricks Runtime 16.4

Databricks supports both 2.12 and 2.13 scala versions for 16.4, ensure the correct runtime is used.

2.1.2 Testing out kogito-4-spark via Notebooks

You can use the appropriate runtime kogito-4-spark_testshade artefact jar (e.g. DBR 16.4) from maven to upload into your workspace / notebook env (or add via maven). When using Databricks make sure to use the appropriate _Version.dbr builds.

Then using:

```
import com.sparkutils.dmn.kogito.tests.TestSuite
import com.sparkutils.dmn.kogito.TestUtils

TestUtils.setPath("path_where_test_files_should_be_generated")
TestSuite.runTests()
```

in your cell will run through all of the test suite used when building kogito-4-spark.

In Databricks notebooks you can set the path up via:

```
val fileLoc = "./kogito-4-spark-testdir"''
TestUtils.setPath(fileLoc)
```

Ideally at the end of your runs you'll see - after 10 minutes or so and some stdout - for example a run on DBR 16.4 provides:

```
Time: 63.686

OK (34 tests)

Finished. Result: Failures: 0. Ignored: 0. Tests run: 34. Time: 633686ms.
import com.sparkutils.dmn.kogito.tests.TestSuite
import com.sparkutils.dmn.kogito.TestUtils
fileLoc: String = ./kogito-4-spark-testdir
```

Last update: May 9, 2025 08:34:16

Created: May 9, 2025 08:34:16

Copyright @ 2025