kogito-4-spark

Kogito run over rows, scaled by Spark

Chris Twiner

Table of contents

1. kogito-4-spark - 0.1.0

verage

Statement	99.10 %	Branch	99.44 %

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

- NEW Spark 4.1, 4.2, Databricks 17.3 and 18 runtime support
- NEW Spark Connect support via dmn-4-spark

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 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. The input type can be left as an empty string and the ddl type of the output expression will be used, JSON must however be explicitly provided.

Static configuration can be passed via uncorrelated subqueries

Although you can use DMNInputField expressions with correlated subqueries to simulate configurable joins, you can also use uncorrelated subqueries to load static configuration data into the input context:

```
DMNInputField("(select collect_set(key) from baseData)","","listConfig"),
DMNInputField("(select map_from_entries( collect_list(struct(string(key), value)) ) from baseData)","","mapConfig"),
```

The first listConfig creates a simple array in the listConfig context entry, suitable for "list contains", whereas mapConfig represents a JavaMap suitable for "get value" calls. All key entries in maps must be strings for "get value" to work, not providing a string will cause an error about the GetValueFunction "get value" not being found.

Uncorrelated subqueries are only loaded once per partition.

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).

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.

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 (accessible via Constants):

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 additional dmnDebugMode and dmnMessages fields, in the case where no issues are present this is likely overkill and should be kept for debug information only. The dmnDebugMode field has the following DDL type (also found in ResultProcessors.debugDDL with KogitoResult provided to serialize it):

```
dmnDebugMode: 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
> >
```

with the dmnMessages field having ddl of (also found in ResultProcessors.messagesDDL with the KogitoMessage type provided to serialize it):

```
dmnMessages: array< struct<
    sourceId: String,
    sourceReference: String,
    exception: String,
    feelEvent: struct<
      severity: String,
      message: String,
      line: Integer,
      column: Integer,
      sourceException: String,
      offendingSymbol: 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"}]
```

---

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: August 13, 2026 18:25:43

Created: August 13, 2026 18:25:43

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 Running on Databricks Runtime 17+

Databricks 17+ supports the base Spark 4 dataset api, allowing both Connect and Classic usage from the same code base.

The following test combinations are supported as of 0.1.0:

Compute Type	Cluster Library	Extension	Connect Via dmn-4-spark api	SPARKUTILS_DISABLE_CLASSI (default false)
Non Shared	kogito-4-spark_testshade_18.3.dbr			
Non Shared	kogito-4-spark_testshade_18.3.dbr	kogito-4-spark_testshade_18.3.dbr	☑	
Shared Compute	kogito-4-spark_connect_testshade_18.3.dbr	kogito-4-spark_testshade_18.3.dbr	☑	true
Shared Compute	kogito-4-spark_connect_testshade_4.1.0.oss	kogito-4-spark_testshade_18.3.dbr	☑	true
Shared Compute	dmn-4-spark_connect_api_18.3.dbr	kogito-4-spark_18.3.dbr	☑	
Shared Compute	dmn-4-spark_connect_api_4.1.0.oss	kogito-4-spark_18.3.dbr	☑	

Databricks 18 has been tested as of release__18.3.x-snapshot-photon-scala2.13__databricks__18.3.3__bfd1bbf__b0356d3__jenkins__bc102bd__format-3 (search for spark.databricks.clusterUsageTags.sparkImageLabel on the environment spark properties page for the exact version your cluster uses). Similarly, 19 works with 18.3 artifacts as of release__19.x-snapshot-photon-scala2.13__databricks__19.2.4__affc832__32d3219__jenkins__59c4a83__format-3, ymmv on later releases and a full 19.dbr release may be needed.

2.1.3 Testing out kogito-4-spark via Notebooks

You can use the appropriate runtime kogito-4-spark_testshade artefact jar (e.g. DBR 18.3) 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.DMNTestRunner
import com.sparkutils.testing.SparkTestUtils

// uncomment to disable connect test usage on runtimes that support it, like DBR 17.3
// System.setProperty("SPARKUTILS_DISABLE_CONNECT_TESTS","true")
```

```
// uncomment to disable classic test usage on runtimes that support connect, DBR 17.3
// a good use case is simulating a UC shared cluster (with init script / spark session extensions enabled)
// when running on a classic cluster setup.
// On an actual UC Shared Compute cluster this is true by default (as the connect.SparkSession is provided)
// System.setProperty("SPARKUTILS_DISABLE_CLASSIC_TESTS","true")

// for running on azure set the configuration for both classic and connect client
val keyMap = Map(s"fs.azure.account.key.${srv_path}${dfs}" -> accountKey)
SparkTestUtils.setRuntimeConnectClientConfig(keyMap)
SparkTestUtils.setRuntimeClassicConfig(keyMap)

val root_path = loc
SparkTestUtils.setPath(root_path+"/kogito")
DMNTestRunner.test()
```

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

Ideally, at the end of your runs you'll see - after 2 minutes or so and some stdout - for example a run on DBR 18.3 provides:

```
Run completed in 1 minute, 41 seconds.
Total number of tests run: 82
Suites: completed 5, aborted 0
Tests: succeeded 82, failed 0, canceled 0, ignored 0, pending 0
All tests passed.
Kogito4Spark - gc'ing after finishing test batch 0
all Kogito4Spark test batches completed
```

The exact number of tests run depends on the test setup and connect usage (82 are expected for standard classic clusters, 79 for connect).

2.1.4 Spark Connect Session Extensions

Starting with Spark 4 and dmn-4-spark 0.1.0 the api is now a separate jar that can be used remotely over Spark 4 Connect.

In order to run against remote clusters the configuration option:

```
spark.sql.extensions=com.sparkutils.dmn.DMN4SparkExtension
```

must be used. In Databricks this also requires using an init script to copy the implementation jars e.g.:

```
#!/bin/bash

cp /Volumes/databricks_ws/default/jars/kogito-4-spark_testshade_4.1.0.oss_4.1_2.13-0.1.0.jar /databricks/jars/kogito-4-
spark_testshade_4.1.0.oss_4.1_2.13-0.1.0.jar
```

When using this approach on a 'standard' shared Databricks cluster the cluster jar must be the _connect versions, based on the dmn-4-spark_api only and not the backend code.

Last update: August 13, 2026 18:25:43

Created: August 13, 2026 18:25:43

3. About

3.1 Changelog

3.1.1 0.1.0 Spark 4.1 DBR 18.3 Xth August, 2026

All Databricks EOS runtimes are removed.

#19 - Support for Spark 4.1, Databricks 17.3 and 18.3 (Runtime 18) is added

#21 - Spark 4.2 and Connect support added

dmn-4-spark #9

kogito-4-spark now provides two testshades, the original, as with 0.0.1 versions, provides the full set of functionality and the _connect version simply provides the test cases and the dmn-4-spark api.

dmn-4-spark #12 - DMNExpressions move to non-deterministic, reducing overhead and increasing correctness for plans re-using projections

3.1.2 0.0.1 Initial Version Xth May, 2025

Initial implementation of the dmn-4-spark api, providing:

- Arbitrary nested structure handling
- JSON input and output types
- Support for DDL DMNResult conversion including status output
- DMNInputFields can have non JSON type derived based on the fieldExpression
- Context input null handling is configurable
- WholestageCodegen support
- Kogito 10.2 support

Last update: August 13, 2026 18:25:43
Created: August 13, 2026 18:25:43

4. Scala Docs

As of 0.1.0 the api is split from the backend implementation.

The utility and result interface can be found here.

The server side interface, typically unnecessary for most users, can be found here.

Last update: August 13, 2026 18:25:43

Created: August 13, 2026 18:25:43