column-lineage

October 11, 2022

1 Column Lineage Demo - October 2022

In this document we provide a demo of current implementation of Column Level Lineage within Marquez and Openlinege

1.1 Envirnonment setup

- I've tested this demo on `Mac 12.6` with `Docker Desktop 4.12.0`.
- I've setup Spark Juper environment like described here here
- I've build and run Marquez locally because I wanted to work on master branch (requires Java 17):

```
./gradlew clean build
./gradlew :api:runShadow
```

Once an October version of Marquez is released, testing this with Marquez docker image may be an easier option.

Let's check first if Marquez instance is properly running under a defined address. Returned status code should be 200.

[42]:
```
import json,requests
marquez_url = "http://host.docker.internal:8080" ## this may depend on your␣
 ↪local setup
if (requests.get("{}/api/v1/namespaces".format(marquez_url)).status_code ==␣
 ↪200):
    print("Marquez is OK.")
else:
    print("Cannot connect to Marquez")
```

```
Marquez is OK.
```

If Marquez connection is OK, we can start Spark context with OpenLineage pointed to Marquez

[43]:
```
from pyspark.sql import SparkSession

spark = (SparkSession.builder.master('local')
         .appName('sample_spark')
         .config('spark.extraListeners', 'io.openlineage.spark.agent.
 ↪OpenLineageSparkListener')
```

```
        .config('spark.jars.packages', 'io.openlineage:openlineage-spark:0.15.
↪1')
        .config('spark.openlineage.url', '{}/api/v1/namespaces/column-lineage/
↪'.format(marquez_url))
        .getOrCreate())
```

Let's clear docker's existing warehouse.

[44]:
```
%rm -rf /home/jovyan/notebooks/spark-warehouse/*
```

1.2 Run example Spark Job

Let's create now four datasets: `dataset_a`, `dataset_b`, `dataset_c`, `dataset_d`: * `dataset_a` has to columns `col_1` and `col_2` filled with some data, * `dataset_b` has one column `col_3` and is created from `dataset_a`, * `dataset_c` with `col_4` and `dataset_d` with `col_5` are created from `dataset_b`

[45]:
```
spark.createDataFrame([
    {'col_1': 1, 'col_2': 2},
    {'col_1': 3, 'col_2': 4}
]).write.mode("overwrite").saveAsTable('dataset_a')
spark.sql("SELECT col_1 + col_2 AS col_3 FROM dataset_a").write.
↪mode("overwrite").saveAsTable('dataset_b')
spark.sql("SELECT col_3 AS col_4 FROM dataset_b").write.mode("overwrite").
↪saveAsTable('dataset_c')
spark.sql("SELECT col_3 AS col_5 FROM dataset_b").write.mode("overwrite").
↪saveAsTable('dataset_d')
```

```
22/10/11 15:46:09 WARN HadoopFSUtils: The directory
file:/home/jovyan/notebooks/spark-warehouse/dataset_c was not found. Was it
deleted very recently?
22/10/11 15:46:10 WARN HadoopFSUtils: The directory
file:/home/jovyan/notebooks/spark-warehouse/dataset_d was not found. Was it
deleted very recently?
```

This should result in following column lineage graph: * `col_3` is created out of `col_1` and `col_2`, * `col_4` and `col_5` depend on `col_3`.

1.3 Marquez API

1.3.1 Get dataset resource with column lineage included

First we may list some example datasets:

[46]:
```
datasets = requests.get("{}/api/v1/namespaces/file/datasets".
↪format(marquez_url)).json()
print(json.dumps(datasets["datasets"][0]["id"], indent=2))
```

```
{
  "namespace": "file",
  "name": "/home/jovyan/notebooks/spark-warehouse/dataset_a"
}
```


Let's try now to fetch a specific dataset: * **namespace**: `file`, * **name**: `/home/jovyan/notebooks/spark-warehouse/dataset_c`

We need to encode dataset name to be able to pass it through URL.

[9]:
```
import urllib
encoded_name = urllib.parse.quote_plus("/home/jovyan/notebooks/spark-warehouse/
 ↪dataset_c")
```

`dataset_c` was created from a single column `col_3` in `dataset_b`, so its column lineage section should only contain a single field.

[47]:
```
dataset = requests.get("{}/api/v1/namespaces/file/datasets/{}".
 ↪format(marquez_url, encoded_name)).json()
print(json.dumps(dataset["columnLineage"], indent=2))
```


```
[
  {
    "name": "col_4",
    "inputFields": [
      {
        "namespace": "file",
        "dataset": "/home/jovyan/notebooks/spark-warehouse/dataset_b",
        "field": "col_3"
      }
    ],
    "transformationDescription": null,
    "transformationType": null
  }
]
```


Fields `transformationDescription` and `transformationType` are available in the OpenLineage standard specification but not implemented in Spark integration (which is the only one).

Column lineage within dataset resource does not return a whole column lineage graph. This is a desired behaviour as a separate column lineage endpoint is intended to fetch furter dependencies.

1.3.2 Get column lineage graph

Column-lineage endpoint returns a lineage graph by specified starting point (`nodeId`) and depth which is 20 by default. Starting point can be a dataset field or a dataset where dataset is equivalent to adding all dataset fields as starting points:

- `nodeId=dataset:some-namespace:some-dataset`
- `nodeId=datasetField:some-namespace:some-dataset:some-Field`

We can distinguish **upstream** and **downstream** lineages. The endpoint by default returns only **upstream** lineage. In other words, it returns all columns that were used to produce a requsted fields while omitting fields that were produced by requested fields.

Some datasets and fields can be used by hundreds or thousands of jobs, so we decided to avoid sending this information by default. This can be achieved with an extra request param `withDownstream` presented later.

Let's check now the column lineage graph of `col_4` in `dataset_c`:

[48]:
```
print(json.dumps(requests.get(
    "{}/api/v1/column-lineage?nodeId=datasetField:file:{}:col_4".
 ↪format(marquez_url, encoded_name)
).json(), indent=2))
```


```
{
  "graph": [
    {
      "id": "datasetField:file:/home/jovyan/notebooks/spark-
warehouse/dataset_a:col_1",
      "type": "DATASET_FIELD",
      "data": null,
      "inEdges": [],
      "outEdges": [
        {
          "origin": "datasetField:file:/home/jovyan/notebooks/spark-
warehouse/dataset_a:col_1",
          "destination": "datasetField:file:/home/jovyan/notebooks/spark-
warehouse/dataset_b:col_3"
        }
      ]
    },
    {
      "id": "datasetField:file:/home/jovyan/notebooks/spark-
warehouse/dataset_a:col_2",
      "type": "DATASET_FIELD",
      "data": null,
      "inEdges": [],
      "outEdges": [
        {
          "origin": "datasetField:file:/home/jovyan/notebooks/spark-
warehouse/dataset_a:col_2",
          "destination": "datasetField:file:/home/jovyan/notebooks/spark-
warehouse/dataset_b:col_3"
        }
      ]
    },
    {
      "id": "datasetField:file:/home/jovyan/notebooks/spark-
```

```
warehouse/dataset_b:col_3",
      "type": "DATASET_FIELD",
      "data": null,
      "inEdges": [
        {
          "origin": "datasetField:file:/home/jovyan/notebooks/spark-
warehouse/dataset_b:col_3",
          "destination": "datasetField:file:/home/jovyan/notebooks/spark-
warehouse/dataset_a:col_1"
        },
        {
          "origin": "datasetField:file:/home/jovyan/notebooks/spark-
warehouse/dataset_b:col_3",
          "destination": "datasetField:file:/home/jovyan/notebooks/spark-
warehouse/dataset_a:col_2"
        }
      ],
      "outEdges": [
        {
          "origin": "datasetField:file:/home/jovyan/notebooks/spark-
warehouse/dataset_b:col_3",
          "destination": "datasetField:file:/home/jovyan/notebooks/spark-
warehouse/dataset_c:col_4"
        }
      ]
    },
    {
      "id": "datasetField:file:/home/jovyan/notebooks/spark-
warehouse/dataset_c:col_4",
      "type": "DATASET_FIELD",
      "data": {
        "type": "DATASET_FIELD",
        "namespace": "file",
        "dataset": "/home/jovyan/notebooks/spark-warehouse/dataset_c",
        "field": "col_4",
        "fieldType": "long",
        "transformationDescription": null,
        "transformationType": null,
        "inputFields": [
          {
            "namespace": "file",
            "dataset": "/home/jovyan/notebooks/spark-warehouse/dataset_b",
            "field": "col_3"
          }
        ]
      },
      "inEdges": [
        {
```

```
          "origin": "datasetField:file:/home/jovyan/notebooks/spark-
warehouse/dataset_c:col_4",
          "destination": "datasetField:file:/home/jovyan/notebooks/spark-
warehouse/dataset_b:col_3"
        }
      ],
      "outEdges": []
    }
  ]
}
```


It contains `dataset_b:col_3`, `dataset_a:col_1` and `dataset_a:col_2` as nodes, while `dataset_d` is not not returned as it is unrelated to column lingea of the requested node.

Response is returned in a form of `Lineage` graph, the same as existing `lineage` endpoint in Marquez. It contains dataset fields as graph nodesm, each with data section containing node's information and edges attached.

1.3.3 Downstream column lineage

Lineage endpoint returns by default only upstream lineage. In order to fetch downstream, an extra parameter `withDownstream=true` has to be added.

We will test it on `dataset_b` and verify that only upstream columns `dataset_a:col_1`, `dataset_a:col_2` are returned, but also `dataset_c:col_4` and `dataset_d:col_5`.

[41]:
```
dataset_c_encoded_name = urllib.parse.quote_plus("/home/jovyan/notebooks/
↪spark-warehouse/dataset_c");

# print(json.dumps(requests.get(
#     "{}/api/v1/column-lineage?nodeId=datasetField:file:{}:
↪col_4&withDownstream=true"
#     .format(marquez_url, dataset_c_encoded_name)
# ).json(), indent=2))
```

Alert: Downstream lineage has a bug. The same edge is being traversed up and down resulting in multiple input fields for `dataset_b:col_3`. Still work in progress.

This ends the demo.