# RunEvent-less Metadata Emission

**Summary:** Technical proposal for creating Dataset and Job metadata emissions without a RunEvent

**Created:** Mar 7, 2022
**Updated:** Mar 7, 2023
**Status:** WIP
**Author:** Maciej Obuchowski
**Contributors:** Julien Le Dem , Benji Lampel



# Context

PRD: Emitting dataset and job metadata outside of RunEvent

# Implementation

## Backward compatibility considerations

Current OpenLineage events are explicitly RunEvents with an eventType="START|COMPLETE|FAIL|..."
Mixing Job and Dataset events in the same stream creates potential issues.

Option 1: We attempt to make events backwards compatible by adding new events in the same stream. There would be some mechanism (field based for example) to differentiate Dataset/Job events from run events. We'd assume the consumers ignore the events they

don’t understand. This is a strong assumption - it will be backwards incompatible for consumers that do not follow this.

Option 2: We create a V2 stream in the protocol (new http endpoint / kafka topic), and start with a blank slate of events with a clear contract on how we can add more events in the future. There needs to be a transition plan from the V1 protocol to V2.

# Backward incompatible option (separate V2 event stream)

- Option 1: Event model will be extended into:
  {
  runEvents: [ array of RunEvent ]
  datasetEvents: [ array of DatasetEvent]
  …
  }
  The advantages of this approach:
  - There is no hidden logic or dependencies in event fields determining content and context of the event.
  - It’s clean for the new users.
  - It’s extendable to include other event types in future (we still have no clear way how to include BI dashboards or data consumer applications in lineage landscape).

  The disadvantages are:
  - It’s a breaking change.

- seamless transition from V1 to V2

## Migration plan for consumers

# Backward compatible options

To facilitate this, we’d want to emit different events, constituting only a single dataset and associated facets. Let’s call it DatasetEvent, in contrast to the already existing RunEvent.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://openlineage.io/spec/2-0-0/DatasetEvent.json",
  "$defs": {
    "DatasetEvent": {
      "properties": {
        "dataset": {
          "$ref": "https://openlineage.io/spec/1-0-2/OpenLineage.json#/$defs/Dataset"
        },
        "producer": {
```

```
            "description": "URI identifying the producer of this metadata. For example this could be a git url with a given tag or sha",
            "type": "string",
            "format": "uri",
            "example": "https://github.com/OpenLineage/OpenLineage/blob/v1-0-0/client"
          },
          "schemaURL": {
            "description": "The JSON Pointer (https://tools.ietf.org/html/rfc6901) URL to the corresponding version of the schema definition for this RunEvent",
            "type": "string",
            "format": "uri",
            "example": "https://openlineage.io/spec/0-0-1/OpenLineage.json"
          }
        },
        "required": [
          "dataset",
          "producer",
          "schemaURL"
        ]
      ]
    }
}
```

## JobEvent:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://openlineage.io/spec/2-0-0/JobEvent.json",
  "$defs": {
    "JobEvent": {
      "properties": {
        "job": {
          "$ref": "https://openlineage.io/spec/1-0-2/OpenLineage.json#/$defs/Job"
        },
        "producer": {
          "description": "URI identifying the producer of this metadata. For example this could be a git url with a given tag or sha",
          "type": "string",
          "format": "uri",
          "example": "https://github.com/OpenLineage/OpenLineage/blob/v1-0-0/client"
        },
        "schemaURL": {
          "description": "The JSON Pointer (https://tools.ietf.org/html/rfc6901) URL to the corresponding version of the schema definition for this RunEvent",
          "type": "string",
          "format": "uri",
          "example": "https://openlineage.io/spec/0-0-1/OpenLineage.json"
        }
      },
      "required": [
        "job",
        "producer",
        "schemaURL"
      ]
    }
  }
```

}

## Backend

Additional event type needs to be understood by OpenLineage backends.
Let's take a look at HTTP transport.
There are few ways to add support for new event types for it. First, is to reuse existing endpoint `POST /api/v1/lineage`
In a way, it adds simplicity to the model - single endpoint. On the other hand, it adds complexity on the backend side. It has to understand which event it's deserializing.

Reference implementation - Marquez uses Jackson library to handle JSON. Deserializing different types can be done by custom deserializer, but the recommended way to deserialize different types in one endpoint is via polymorphic deserialization:
https://github.com/FasterXML/jackson-docs/wiki/JacksonPolymorphicDeserialization
This requires distinguishing data types by some field with fixed name, for example

```
@JsonTypeInfo(
  use = JsonTypeInfo.Id.CUSTOM,
  include = As.NONE,
  property = "schemaURL")
@JsonTypeIdResolver(OpenlineageEventResolver.class)
@JsonSubTypes({
   @JsonSubTypes.Type(value = RunEvent.class, name = "runEvent"),
   @JsonSubTypes.Type(value = DatasetEvent.class, name = "datasetEvent")
})
```

The great thing is that we have such a field that we can use to distinguish types - schemaURL. Different event types would have different type names. Notice use of JsonTypeInfo.Id.CUSTOM and OpenlineageEventResolver - this is necessary due to version numbers in schemaURL. Without it, only the exact match is performed by Jackson.

Moreover, HTTP transport isn't the only transport we want to support. The same problem will happen on the Kafka side - having two types of events in one topic isn't very easy. The solutions are actually very similar - either use two topics, which fragments metadata, and introduces ordering problems, or have them in one and have a way to correctly deserialize.

To sum up, considering we properly utilize schemaURL fields, I believe we should use the same endpoint and kafka topic by default. However, this is an important topic and it would be good to see different opinions.

Some articles that go over it from Kafka perspective:
https://www.confluent.io/blog/put-several-event-types-kafka-topic/

https://www.confluent.io/blog/multiple-event-types-in-the-same-kafka-topic/

## Where to connect the facet?

This also touches on a point: how does metadata send DatasetEvent (or JobEvent) should be understood by consumers. Julien suggested:

The current model is as follows:
Jobs:
- run facet: only applies to a given run of a job. (example: ExternalQueryRunFacet) But does not apply to subsequent runs.
- Job facet: applies to the job in general, the latest replaces the current value for the job and applies to future versions until replaced. (example: OwnershipJobFacet)
Datasets:
- input/output facets: only applies to a given run. (ex: DataQualityMetricsInputDatasetFacet or OutputStatisticsOutputDatasetFacet) but not to subsequent runs
- DatasetFacet: applies to the dataset in general, the latest replaces the current value for the Datasets and applies to future versions until replaced. (example: OwnershipDatasetFacet)

The same should apply for JobEvent, but simpler: it does only allow Job metadata to be emitted and Dataset not.