Beam Protobuf Schema (Java)

self-link: https://s.apache.org/beam-protobuf

Authors: baeminbo@google.com

Contributors:

Reviewers:

Status: Review

Last revised: 2025-06-05

Visibility: Public

Objective

This proposal aims to correctly align nullability between Protobuf and Apache Beam Schemas. Currently, optional fields in both Proto2 and Proto3 are converted to non-nullable field types within Beam's `ProtoSchemaTranslator`. We'll update this conversion rule to ensure proper nullability is maintained.

Conversion Rules

Scalar Value Types

Scalar value types are primitive types in Protobuf.

Proto Type	**Java Type (Protobuf)**	**Field Type (Beam)**	**Java Type (Beam)**
double	double	FieldType.DOUBLE	double
float	float	FieldType.FLOAT	float

int32	int	FieldType.INT32	int
int64	long	FieldType.INT64	long
uint32	int[1]	ProtoSchemaLogicalTypes.UInt32	int
uint64	long[1]	ProtoSchemaLogicalTypes.UInt64	long
sint32	int	ProtoSchemaLogicalTypes.SInt32	int
sint64	long	ProtoSchemaLogicalTypes.SInt64	long
fixed32	int[1]	ProtoSchemaLogicalTypes.Fixed32	int
fixed64	long[1]	ProtoSchemaLogicalTypes.Fixed64	long
sfixed32	int	ProtoSchemaLogicalTypes.SFixed32	int
sfixed64	long	ProtoSchemaLogicalTypes.SFixed64	long
bool	boolean	FieldType.BOOLEAN	boolean
string	String	FieldType.String	String
bytes	ByteString	FieldType.BYTES	byte[]

Field Cardinality

- `required` (proto2) and `implicit` (proto3): converted to **non-null** field type in Beam
- `optional`: converted to **nullable** field type in Beam.
- `repeated <type>`: converted to a **non-null** FieldType.ARRAY with the **non-null** element type for the protobuf type `<type>`.

For example,

```
Protobuf
int64 id = 2;
```

```
optional string title = 3;
repeated string messages = 4;
```

```None
id [INT64 NOT_NULL]
title [STRING NULLABLE]
messages [ARRAY NULLABLE]:
  <element_type> [STRING NOT_NULL]
```

OneOf

Protobuf fields contained in an `oneof` are converted to nested **nullable** fields in a **nullable** logical type OneOfType.

For example,

```Protobuf
oneof user {
  int32 id = 3;
  string name = 7;
}
```

```None
user [OneOfType NULLABLE]:
  id   [INT32 NULLABLE]  // case enum: 3
  name [STRING NULLABLE] // case enum: 7
```

Map

A Map field in Protobuf is converted to a **non-null** FieldType.Map with **non-null** key and value types. The key type can be integral or string type. Thus, any scala value types except `double`, `float` and `bytes`.

For example,

```
Protobuf
map<int32, string> metadata = 3;
```

```
None
metadata [MAP NULLABLE]:
  <key>   [INT32 NOT_NULL]
  <value> [STRING NOT_NULL]
```

Enum

An enum value is converted to EnumerationType and it's **nullable** if `optional.`

For example,

```
Protobuf
enum ErrorCode {
  OK = 0;
  INVALID = 4;
  INTERNAL = 5;
}

ErrorCode code = 1;
```

```
None
code [ENUM NOT_NULL]:
  OK = 0
  INVALID = 4
  INTERNAL = 5
```

Message

A message type is converted to a **nullable** FieldType.ROW.

Well-Known Messages

Certain messages in `google.protobuf` package are handled uniquely; they aren't converted to FieldType.ROW.

Wrapper messages for scala value types:

- BoolValue → **nullable** FieldType.BOOLEAN
- BytesValue → **nullable** FieldType.BYTES
- DoubleValue → **nullable** FieldType.DOUBLE
- FloatValue → **nullable** FieldType.FLOAT
- Int32Value → **nullable** FieldType.INT32
- Int64Value → **nullable** FieldType.INT64
- StringValue → **nullable** FieldType.STRING
- UInt32Value → **nullable** ProtoSchemaLogicalTypes.UInt32
- UInt64Value → **nullable** ProtoSchemaLogicalTypes.UInt64

Other messages:

- Any → Not supported.
- Duration → **nullable** logicaltypes.NanosDuration
- Timestamp → **nullable** logicaltypes.NanosInstant

Extension (proto2)

As of version 2.65.0, Protobuf extension fields are not supported in Beam's Protobuf Schema. This is because extensions are not part of Protobuf descriptor, which our schema inference relies on. Implementing support would be a significant and breaking change. Support for extensions will not be included as part of this work. The extensions in Protobuf Descriptor and messages are **discarded** during the conversion process.

Custom Default Scala Values (proto2)

The `default` keyword option to override default scala values is **ignored**.

Unknown Fields

If a Protobuf message has unknown fields, the field values are **discarded** when converted to a Beam Row.

Breaking Changes from v2.65.0

1. `optional` fields in Protobuf will be converted to **nullable** fields. Previously, they were converted to **non-null** fields. Due to this change, a single **nullable** field type in Beam can represent two different Protobuf types. For example, nullable FieldType.INT32 in Beam can be `optional int32` or `google.protobuf.Int32Value` in Protobuf.
2. The OneOfType field for Protobuf `oneof` will be **nullable**. Previously, they were **non-null**. There's no change to the nested types within OneOfType; they will remain **nullable**, as they were before.

Implementation

Three existing classes need to be changed.

ProtoSchemaTranslator

The `getSchema` should be modified to apply the changes.

ProtoDynamicMessageSchema

This is a class to convert between Beam Row and Protobuf DynamicMessage.

Currently, `ProtoDynamicMessageSchema` converts between Protobuf messages and Beam Rows using Beam Schema only. However, as a single Beam field type can represent two Protobuf types, it needs Protofile descriptor information for correct conversion.

Conversion from Protobuf Message to Beam Row:

1. Non-null Beam fields get their value from the Protobuf value or the default value of the scalar type (e.g., 0 for `int32`, "" for `string`, false for `bool`)
2. Nullable Beam fields get null if the Protobuf value is not present (e.g., an unset set optional field, map, repeated or message type). Otherwise, nullable Beam fields get their value from the Protobuf value

Conversion from Beam Row to Protobuf Message:

1. Protobuf fields without presence (e.g., a required or implicit) get their value from the Beam value or the default value if the Beam value is `null`.
2. Protobuf fields with presence (e.g., an optional, map, repeated, message type)gets their value from the Beam value or `null` if the Beam value is `null`.

Since `ProtoDynamicMessageSchema` is used to parse Protobuf options within `ProtoSchemaTranslator`, any updates to the `ProtoDynamicMessageSchema` constructor will need a corresponding change in `ProtoSchemaTranslator`.

~~For backward-compatibility of `ProtoDynamicMessageSchema`, currently relying on Beam Schema only for conversion, we might need to implement a method in `ProtoSchemaTranslator.getDescriptor(Schema)` for generating Protobuf Descriptor from Beam Schema. This will be based on the conversion rule in v2.65.0 for backward compatibility. The generated Descriptor will be used for `ProtoDynamicMessageSchema.forSchema(Schema)`~~
The `ProtoDynamicMessageSchema.forSchema(Schema)` will be removed as its only use was for parsing Protobuf options within `ProtoSchemaTranslator.getSchema(Descriptor)`. Thus, we can use Protobuf Descriptor and Beam Schema together for conversion.

Due to the class complexity, a new class `ProtoBeamConverter` is created, and this class will be deprecated. The `getToRowFunction()` and `getFromRowFunction()` in `ProtoDynamicMessageSchema` are routed to `toRow` and `toProto` in `ProtoBeamConverter`.

ProtoMessageSchema

This is a class to convert between Beam Row and a specific Protobuf Message class, using ByteBuddy technology intensively.

Since `ProtoMessageSchema` already contains the necessary Protobuf Message class details, implementing this within `ProtoMessageSchema` would require fewer changes than with `ProtoDynamicMessageSchema`.

ProtoBeamConverter

A new `ProtoBeamConverter` is created to replace `ProtoDynamicSchema`, which has two static methods: `toProto` and `toRow`

```java
Class ProtoBeamConverter {
    /** Returns a conversion method from Beam Row to Protobuf Message. */
  public static SerializableFunction<@NonNull Row, @NonNull Message>
toProto(Descriptors.Descriptor descriptor);

    /** Returns a conversion method from Protobuf Message to Beam Row. */
  public static SerializableFunction<@NonNull Message, @NonNull Row>
toRow(Schema schema);
}
```

The `toProto` nullability rule:

Protobuf Presence	**Beam Value**	**Proto Value**
Yes (optional, message, etc.)	null	null
Yes	non-null	non-null
No (implicit, required)	null	default
No	non-null	non-null

The `toRow` nullability rule:

Beam FieldType	**Proto Value**	**Beam Value**
Nullable	null	null
Nullable	non-null	null
Non-null	null	default[4]
Non-null	non-null	non-null

The conversion methods match source and destination fields based on their **name**s. The Protobuf tag number option (`beam:option:proto:meta:number`) in each Beam field is not used in this matching process. This fixes the bug where the field order is shuffled in Beam Schema. See A Bug when Schema Field Order is Changed.

As mentioned at Extension (proto2) and Unknown Fields, the fields not listed in Proto Descriptor and Beam Schema are discarded in `toProto(Descriptor)` and `toRow(Schema)`, respectively. If the value type is invalid (e.g., Integer → Long), it will throw an exception in building the Protobuf Message and Beam Row.

The enum names are not used in conversion. The value can be an unrecognized value in Protobuf and Beam. However, this is allowed in Protobuf and Beam. For example, consider the following `Enum` with two elements: `ZERO(0)` and `ONE(1)`. The enum value 10 is not recognized in the enum, but valid value in Protobuf and Beam.

```
Protobuf
enum Enum {
  ZERO = 0;
  ONE = 1;
}

message EnumMessage {
  Enum enum = 1;
}
```

```
None
Message: { enum: 10 }
<==>
Row: { enum: 10 }
```

Appendix

BigQuery to Beam Schema

See BigQueryIO for conversion between BigQuery types and Java types, and BigQueryUtils between BigQuery types and Beam field types. BigQuery fields whose mode is NULLABLE are converted to **nullable** Beam field types.

BigQuery Standard SQL	**Avro Type**	**Java Type**	**Field Type (Beam)**
BOOLEAN	boolean	Boolean	FieldType.BOOLEAN
INT64	long	Long	FieldType.INT64
FLOAT64	double	Double	FieldType.DOUBLE
BYTES	bytes	java.nio.ByteBuffer	FieldType.BYTES
STRING	string	CharSequence	FieldType.STRING
DATE	int	Integer	SqlTypes.Date (= logicaltypes.Date)
DATETIME	string	CharSequence	SqlTypes.DateTime (= logicaltypes.DateTime)
TIMESTAMP	long	Long	SqlTypes.Timetamp (= logicaltypes.MicroInstant)
TIME	long	Long	SqlTypes.Time (= logicaltypes.Time)
NUMERIC	bytes	java.nio.ByteBuffer	FieldType.DECIMAL

BigQuery Standard SQL	Avro Type	Java Type	Field Type (Beam)
GEOGRAPHY	string	CharSequence	FieldType.STRING
ARRAY[2]	array	java.util.Collection	**non-null** FieldType.ARRAY
STRUCT	record	org.apache.avro.GenericRecord	FieldType.ROW[3]

Iceberg to Beam Schema

See IcebergIO and IcebergUtils for conversion between Iceberg types and Beam field types.

Iceberg Type	Field Type (Beam)
BINARY	FieldType.BYTES
BOOLEAN	FieldType.BOOLEAN
STRING	FieldType.STRING
INTEGER	FieldType.INT32
LONG	FieldType.INT64
STRING	FieldType.STRING
FLOAT	FieldType.FLOAT
DOUBLE	FieldType.DOUBLE
TIMESTAMP	SqlTypes.DATETIME, if TimestampType.shouldAdjustToUTC() is false FieldType.DATETIME, otherwise
DATE	SqlTypes.Date (= logicaltypes.Date)
TIME	SqlTypes.Time (= logicaltypes.Time)
LIST	FieldType.ITERABLE
MAP	FieldType.MAP
STRUCT	FieldType.ROW

A Bug when Schema Field Order is Changed

In version 2.65.0, `ProtoDynamicMessageSchema.getFromRowFunction()` has a bug that occurs when processing Beam Rows whose schema field order does not match the Protobuf descriptor.

For example, consider a Protobuf message named `Address` with two fields, `city` and `street`, in the order. Using a schema having `street` and `city` in the order, a row is created with `street` as "fake street" and `city` as "seattle". Then, the result message of `getFromRowFunction` has the `city` as "fake street" and `street` as "seattle", incorrectly.

Protobuf
```
message Address {
  string street = 1;
  string city = 2;
}
```

Java
```
Descriptors.Descriptor descriptor = Proto3Messages.Address.getDescriptor();
SerializableFunction<Row, DynamicMessage> fromRow = ProtoDynamicMessageSchema
    .forDescriptor(ProtoDomain.buildFrom(descriptor), descriptor)
    .getFromRowFunction();

// Schema with different field order
Schema schema = Schema.builder()
    .addField("city", Schema.FieldType.STRING)
    .addField("street", Schema.FieldType.STRING)
    .build();

Row row = Row.withSchema(schema)
    .addValue("seattle")
    .addValue("fake street")
    .build();

DynamicMessage message = fromRow.apply(row);
System.out.println(message);
```

None
```
street: "seattle"
city: "fake street"
```

Because `ProtoDynamicMessageSchema` retrieves Beam `Row` values by field ID (position) rather than by name, it fails when the `Row`'s schema order doesn't match the Protobuf descriptor. This mismatch leads to runtime troubles, such as type exceptions or incorrectly populated fields. This is the root cause of new unit test failures in `ProtoByteUtilsTest` and `PubsubLiteDlqTest` that appeared after the changes in this document were applied.

---

[1] In Java, unsigned 32-bit and 64-bit integers are represented using their signed counterparts, with the top bit simply being stored in the sign bit.

[2] For BigQuery, an ARRAY is a field whose mode is set to REPEATED.

[3] BigQuery STRUCT fields are converted to FieldType.MAP if `SchemaConversionOptions` has `inferMaps` as `true` and the STRUCT only has "key" and "value" fields. However, this conversion doesn't happen in Apache Beam because the `inferMaps` setting is always `false`.

[4] The default value is 0 for numeric values, empty string, empty bytes, empty list, empty map and empty row.

If a Beam FieldType is non-null OneOfType, it cannot convert to a Beam value in `ProtoBeamConverter.toRow` if the source Protobuf message has null for all the proto fields contained in the Oneof, because it cannot decide the default value. This will throw a runtime exception. However, this case doesn't happen in real life because OneOfType must be **nullable**. See OneOf.