KIP-481: SerDe Improvements for Connect Decimal type in JSON

Status

Current state: *Accepted*

Discussion thread: *KIP-481 Discussion*

JIRA: *here*

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

Most JSON data that utilizes precise decimal data represents it as a decimal number. Connect, on the other hand, only supports a binary BASE64 string encoding (see example below). This KIP intends to support both representations so that it can better integrate with legacy systems (and make the internal topic data easier to read/debug):

- serialize the decimal field "foo" with value "10.2345" with the BASE64 setting: {"foo": "D3J5"}
- serialize the decimal field "foo" with value "10.2345" with the NUMERIC setting: {"foo": 10.2345}

Public Interfaces

Introduce a new configuration to the JsonConverter named **decimal.format** to control whether source converters will serialize decimals in numeric or binary formats. The value will be case insensitive and can be either "BASE64" (default, to maintain compatibility) or "NUMERIC".

Proposed Changes

We propose the following changes:

1. Define a new **decimal.format** configuration property on JsonConverter to specify the *serialization* format for Connect DECIMAL logical type values with two allowed literals as the configuration property:
   - **(default)** BASE64 specifies the existing behavior of serializing DECIMAL logical types as base64 encoded binary data (e.g. "D3J5" in the example above); and
   - NUMERIC will serialize Connect DECIMAL logical type values in JSON as a number representing that decimal (e.g. 10.2345 in the example above)
2. The JsonConverter deserialization method currently expects only a BinaryNode, but will be changed to also handle NumericNode by calling NumericNode.decimalValue().
3. JsonDeserializer will now default floating point deserialization to BigDecimal to avoid losing precision. This may impact performance when deserializing doubles - a JMH microbenchmark on my local MBP, this estimated about 3x degradation for deserializing JSON floating points. If the connect schema is not the decimal logical type, the JsonConverter will convert this BigDecimal value into the corresponding floating point java object.
4. Configure the JsonConverter for internal topics with `decimal.format=NUMERIC` so that if the DECIMAL types will be serialized in a more natural representation. This is safe since connect internal topics do not currently use any decimal types.

To understand behavior of this configuration with and without schemas, refer to the table below.

Source Schema (Info)	JsonConverter Behavior		JsonDeserializer Behavior
	Schemas Enabled	Schemas Disabled	
DECIMAL (BASE64)	returns DECIMAL logical type	throws DataException	stores BinaryNode (byte[] data)
DECIMAL (NUMERIC)	returns DECIMAL logical type	**returns FLOAT64 (lossy)***	stores NumericNode (BigDecimal data)
FLOAT64 (Legacy)	returns FLOAT64	returns FLOAT64	stores NumericNode (Double data)
FLOAT64 (KIP-481)	returns FLOAT64	returns FLOAT64	**stores NumericNode (BigDecimal data)****

***** previously it was impossible for a sink converter to read decimal data encoded in BASE64 (a DataException was raised when decoding BINARY data without a schema) - after KIP-481, it is possible that a decimal data produced by a source converter with NUMERIC encoding will be deserialized as a float by a sink converter without schemas enabled. The data being read here will be lossy. This is okay because all existing applications will function exactly the same.

****** with this behavior, users will be provided with a different NumberNode than previously (DecimalNode vs. DoubleNode). Since Jackson requires calling a strongly typed method (e.g. floatValue vs. decimalValue) to extract data from a NumericNode, applications relying on JsonDeserializer will not be affected by this change

Compatibility, Deprecation, and Migration Plan

- The JsonConverter's serialization behavior is identical to legacy behavior when using the default configuration (**decimal.format=BASE64**)
- The JsonConverter's deserialization behavior is backwards compatible and can handle data encoded from earlier versions
- To set **decimal.format** to "NUMERIC" in source converters, all sink converters reading the data (and any other downstream consumers of the source converter output topic) must first upgrade code to include the code change from this KIP.
- Note that the Kafka topic will have messages of mixed serialization format after this change. Rolling back sink converters or any other downstream Kafka consumers to code that cannot handle both formats will be unsafe after upgrading source converters to use NUMERIC serialization formats.

Migration Plan. Assuming that the baseline source converter is on Kafka Connect version "A" <= 2.3 and that all sink converters are on Kafka Connect version "B" <= 2.3, and that this change makes it into Kafka Connect version "C" >= 2.4 the upgrade path to utilize this is:

1. Upgrade all sink converters to version "C" or higher and restart the sink converters. No configuration change is needed for the sink converters.
2. Upgrade the source converters to version "C" or higher.
3. If the Connect worker uses the JsonConverter for the key and/or value converters, optionally set the `decimal.format=NUMERIC` for the key and /or value converter and restart the workers.
4. If desired, update any source connector configs that use the JsonConverter for key and/or value converters to use `decimal.format=NUMERIC`.

Rejected Alternatives

- The original KIP suggested supporting an additional representation - base10 encoded text (e.g. `{"asText":"10.2345"}`). While it is possible to automatically differentiate NUMERIC from BASE10 and BASE64, it is not always possible to differentiate between BASE10 from BASE64. Take, for example, the string "12" - this is both a valid decimal (12) and a valid hex string which represents a decimal (1.8). This causes issues because it is impossible to disambiguate between BASE10 and BASE64 without an additional config - furthermore, this makes the migration from one to the other nearly impossible because it would require that all consumers stop consuming and producers stop producing and atomically updating the config on all of them after deploying the new code, or waiting for the full retention period to pass - neither option is viable. The suggestion in the KIP is strictly an improvement over the existing behavior, even if it doesn't support all combinations.
- Encoding the serialization in the schema for Decimal LogicalType. This is good because it means that the deserializer will be able to decode based on the schema and one converter can handle different topics encoded differently as long as the schema is in line. The problem is that this is specific only to JSON and changing the LogicalType is not the right place.
- Creating a new JsonConverter (e.g. JsonConverterV2) that handles decimals using numeric values. This would cause undue code maintenance burden and also may cause confusion when configuring connectors (e.g. what's the difference between the two converters and why are there two?) - naming them properly is a challenge.