KIP-305: Add Connect primitive number converters

Status

Current state: *Accepted*

Discussion thread: *here*

JIRA: *here*

Released: 2.0.0

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

Motivation

Kafka already has serializers and deserializers for primitive numeric values, such as `IntegerSerializer` and `IntegerDeserializer`. Connect doesn't yet have Converter implementations for these primitives, yet it may be useful for some connectors to use numbers as keys or values within records. In these cases, primitive converters are more useful and desirable than having to use the String, JSON, or Avro converters.

Note that primitive integral numbers makes sense for keys, whereas floating point numbers much much less sense. Despite that, this proposal suggests adding Converters for all of Kafka's existing primitive serdes, keeping symmetry between the available options.

Public Interfaces

Add five new implementations of the existing `org.apache.kafka.connect.storage.Converter` interface, which use the corresponding (existing) Kafka serializer and deserializers that already support nulls. All of the Converter implementations will use the specified Connect optional schema when deserializing values to allow for nulls. These classes will also implement the existing `org.apache.kafka.connect.storage.HeaderConverter` interface, which like the existing StringConverter simply delegates to the normal Converter methods. The topic and header names supplied to the converters do not affect serialization or deserialization.

Class name	Deserialized Schema	Uses existing Serializer	Uses existing Deserializer
`org.apache.kafka.connect.storage.ShortConverter`	`Schema.OPTIONAL_INT16`	`org.apache.kafka.common.serialization.ShortSerializer`	`org.apache.kafka.common.serialization.ShortDeserializer`
`org.apache.kafka.connect.storage.IntegerConverter`	`Schema.OPTIONAL_INT32`	`org.apache.kafka.common.serialization.IntegerSerializer`	`org.apache.kafka.common.serialization.IntegerDeserializer`
`org.apache.kafka.connect.storage.LongConverter`	`Schema.OPTIONAL_INT64`	`org.apache.kafka.common.serialization.LongSerializer`	`org.apache.kafka.common.serialization.LongDeserializer`
`org.apache.kafka.connect.storage.FloatConverter`	`Schema.OPTIONAL_FLOAT32`	`org.apache.kafka.common.serialization.FloatSerializer`	`org.apache.kafka.common.serialization.FloatDeserializer`
`org.apache.kafka.connect.storage.DoubleConverter`	`Schema.OPTIONAL_FLOAT64`	`org.apache.kafka.common.serialization.DoubleSerializer`	`org.apache.kafka.common.serialization.DoubleDeserializer`

None of these classes have configuration options. To use them, a Connect worker or converter simply specifies the class names in the `key.converter` and/or `value.converter` configuration property that are already part of Connect.

Proposed Changes

Implement the five new implementations of the Converter and HeaderConverter interfaces, as outlined above. The implementations will use an abstract base class to encapsulate the common functionality.

See the pull request for implementation details.

Compatibility, Deprecation, and Migration Plan

These are new converters can be used by Connect workers or connectors, but none will be used by default and all behave similarly to existing converters.

- No existing connector or worker configurations are affected. The configurations must be changed in order to use these new converters.
- No existing behavior will be affected.

Additionally, Kafka clients and streams applications can already use Kafka's existing serializers and deserializers. These new converters simply wrap those serdes and are therefore compatible with clients and streams applications.

Rejected Alternatives

One alternative considered was to create a single Converter that specified the desired type in configuration properties. This would have been more complicated to configure, so the simpler approach was taken.

A second alternative that was considered was to use one Converter that could dynamically determine the serialized and deserialized form for each key or value. This was rejected since no corresponding Kafka serdes implementation would work the same way.

Third, it was considered to only provide ShortConverter, IntegerConverter, and LongConverter since integral numbers make sense as message keys. In fact, one might argue that only longs make sense as keys, since the shorts and integers have too small a range to be useful as keys. However, the proposal still includes all five Converters (a) to maintain symmetry with the existing Kafka serdes, and (b) to allow them to be used as HeaderConverters. They are very simple classes, so the maintenance cost is minor. A bigger question is whether the additional converters lead users to use them inappropriately, but here the lack of symmetry with existing Kafka serdes might be a bigger concern.