KIP-301: Schema Inferencing for JsonConverter

Status

Current state: *Under Discussion*

Discussion thread: *here*

JIRA: *here*

Motivation

Though there does exist a converter in the connect-json library called "JsonConverter", there are limitations as to the domain of JSON payloads this converter is compatible with on the Sink Connector side when serializing them into Kafka Connect datatypes; When reading byte arrays from Kafka, the JsonConverter expects its inputs to be a JSON envelope that contains the fields "schema" and "payload", otherwise it'll throw a DataException reporting:

> *JsonConverter with schemas.enable requires "schema" and "payload" fields and may not contain additional fields. If you are trying to deserialize plain JSON data, set schemas.enable=false in your converter configuration.*

(when **schemas.enable** is true)

> *JSON value converted to Kafka Connect must be in envelope containing schema*

(when **schemas.enable** is false)

For example, if your JSON payload looks something on the order of:

```
{
 "c1": 10000,
 "c2": "bar",
 "create_ts": 1501834166000,
 "update_ts": 1501834166000
}
```

This will not be compatible for Sink Connectors that require the schema for data ingest when mapping from Kafka Connect datatypes to, for example, JDBC datatypes. Rather, that data is expected to be structured like so:

```
{
 "schema": {
 "type": "struct",
 "fields": [{
 "type": "int32",
 "optional": true,
 "field": "c1"
 }, {
 "type": "string",
 "optional": true,
 "field": "c2"
 }, {
 "type": "int64",
 "optional": false,
 "name": "org.apache.kafka.connect.data.Timestamp",
 "version": 1,
 "field": "create_ts"
 }, {
 "type": "int64",
 "optional": false,
 "name": "org.apache.kafka.connect.data.Timestamp",
 "version": 1,
 "field": "update_ts"
 }],
 "optional": false,
 "name": "foobar"
 },
 "payload": {
 "c1": 10000,
 "c2": "bar",
 "create_ts": 1501834166000,
 "update_ts": 1501834166000
 }
}
```

The "schema" is a necessary component in order to dictate to the JsonConverter how to map the payload's JSON datatypes to Kafka Connect datatypes on the consumer side, and certain Sink Connectors absolutely require a schema in order for them to function (eg: JdbcSinkConnector). Instead of requiring explicitly defined schemas within all json records from every producer that is part of a data pipeline where a Sink Connector will consume its records, the JsonConverter should have the ability to make inferences based on the contents of the JSON data.

Public Interfaces and Proposed Changes

This proposal introduces a new configuration that would be read by all instances of the JsonConverter. In order to activate it, individual connectors can set this as a connector property, or users can set this as a worker property:

```
value.converter.schemas.infer.enable: "true"
```

The default value for this configuration is false, and the original behavior of passing along SchemaAndValue objects with a null schema to downstream sink connector logic is observed. However, if this configuration is set to "true", the JsonNodeType of each JSON node will be used to draw an inference as to the Kafka Connect datatype that most closely resembles that piece of data. For instance, if our data was:

```
{"test1":"hi","test2":"there","test3":12,"test4":12.5,"test5":null,"test6":true,"test7":false,"test8":
["element1","element2"],"test9":[],"test10":[1]}
```

The inferred schema should yield a STRUCT with the following fields:

name	schema
test1	STRING
test2	STRING
test3	INT64
test4	FLOAT64
test5	STRING
test6	BOOLEAN
test7	BOOLEAN
test8	ARRAY{STRING}
test9	ARRAY{STRING}

test10	ARRAY{INT64}

This is achievable by introducing a recursive method, inferSchema(JsonNode jsonValue), which is capable of both 1.) inferring the schema for simple JSON documents, and 2.) making a recursive call to break down JSON documents with complex data types into their constituent parts and Schemas:

JsonConverter.java

```
private Schema inferSchema(JsonNode jsonValue) {
    switch (jsonValue.getNodeType()) {
        case NULL:
            return Schema.OPTIONAL_STRING_SCHEMA;
        case BOOLEAN:
            return Schema.BOOLEAN_SCHEMA;
        case NUMBER:
            if (jsonValue.isIntegralNumber()) {
                return Schema.INT64_SCHEMA;
            }
            else {
                return Schema.FLOAT64_SCHEMA;
            }
        case ARRAY:
            SchemaBuilder arrayBuilder = SchemaBuilder.array(jsonValue.elements().hasNext() ? inferSchema
(jsonValue.elements().next()) : Schema.OPTIONAL_STRING_SCHEMA);
            return arrayBuilder.build();
        case OBJECT:
            SchemaBuilder structBuilder = SchemaBuilder.struct();
            Iterator<Map.Entry<String, JsonNode>> it = jsonValue.fields();
            while (it.hasNext()) {
                Map.Entry<String, JsonNode> entry = it.next();
                structBuilder.field(entry.getKey(), inferSchema(entry.getValue()));
            }
            return structBuilder.build();
        case STRING:
            return Schema.STRING_SCHEMA;
        case BINARY:
        case MISSING:
        case POJO:
        default:
            return null;
    }
}
```

In the current model, the assumption made in the event that a JSON value is not capable of being intelligibly inferenced (null, []) is that the actual schema is a STRING.

Compatibility, Deprecation, and Migration Plan

Since the default value for **schemas.infer.enable** is false, existing instances of JsonConverters will remain unaffected. In order to migrate to using these new capabilities for existing connectors already configured to use the JsonConverter, users will have to send a PUT request to the */connectors/{connector name}/config* endpoint and include this property:

`value.converter.schemas.infer.enable: "true"`

Once the connector properties are updated, users can then feel free to strip away schema data from the upstream producers as they are no longer necessary for unmarshalling the Kafka Connect Schema.

Rejected Alternatives

N/A