KIP-893: The Kafka protocol should support nullable structs

Status

Current state: *Accepted*

Discussion thread: *here*

JIRA: *here*

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

Motivation

The Kafka RPC protocol supports nullable fields but only for a subset of the available types: string, array, records and bytes. structs are not supported at the moment. In some case, having the ability to make an entire struct nullable makes sense.

For instance, in the context of KIP-848, we would like to provide the assignment to a member only if it has changed or null if it has not. The schema would look as follow:

```
    { "name": "Assignment", "type": "Assignment", "versions": "0+", "nullableVersions": "0+", "default": "null",
      "about": "null if not provided; the assignment otherwise.", "fields": [
        { "name": "Error", "type": "int8", "versions": "0+",
          "about": "The assigned error." },
        { "name": "AssignedTopicPartitions", "type": "[]TopicPartitions", "versions": "0+",
          "about": "The partitions assigned to the member that can be used immediately." },
        { "name": "PendingTopicPartitions", "type": "[]TopicPartitions", "versions": "0+",
          "about": "The partitions assigned to the member that cannot be used because they are not released by
their former owners yet." },
        { "name": "MetadataVersion", "type": "int16", "versions": "0+",
          "about": "The version of the metadata." },
        { "name": "MetadataBytes", "type": "bytes", "versions": "0+",
          "about": "The assigned metadata." }
    ]}
```

Proposed Changes

This KIP proposes to add nullable struct support in the Kafka serialization protocol for both regular and tagged fields. Concretely, this means that using the `nullableVersions` and `default` attributes will be allowed on a struct. The `default` attribute, if defined, will only accept `null`.

Public Interfaces

This section describes how nullable structs will be serialized on the wire. The basic idea is to precede the nullable struct by a byte which represents whether the struct is null or present. The format is slightly different depending on whether the field is optional/tagger or not.

Note that non-nullable structs remain serialized as they are today.

Regular Field

A byte (INT8) representing whether the struct is present (1) or null (-1), followed by the bytes (BYTES) of the struct.

We follow the current convention used by other nullable fields which uses -1 when the field is null.

INT8	BYTES
-1 if null; 1 if not null.	The bytes representing the struct.

Optional/Tagged Field

A unsigned variable integer (VARINT) representing whether the struct is present (1) or null (0), followed by the bytes (BYTES) of the struct.

We follow the current convention used by other nullable tagged fields which uses 0 when the field is null. In any cases, it will always use one byte.

VARINT	BYTES
0 if null; 1 if not null.	The bytes representing the struct.

Compatibility, Deprecation, and Migration Plan

The change is backward compatible.

Test Plan

The change will be covered with unit tests.

Rejected Alternatives

None