KIP-568: Support RPC message type 'double'

Status

Current state: Under discussion

Discussion thread: TODO

JIRA: KAFKA-9474

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

Motivation

The Kafka RPC message format has no inherent support for handling `Double` values, however some interfaces may benefit from their addition. The particular motivating case is in implementing a quotas admin client API (KIP-546), which natively accepts `Double` values, but there's no direct way to define them in the RPC messages. Instead of attempting to work around this limitation, it'd be useful to extent the RPC format to support `Double` values.

Public Interfaces

Adds ***`double`*** as a possible RPC protocol field type.

This has no immediate impact to any existing public interfaces. It'll permit new and existing messages to add `double` fields to their structure in subsequent iterations.

Proposed Changes

Adds the following code block to `clients/src/main/java/org/apache/kafka/common/protocol/types/Type.java` and support code to `Struct.java`.

Note that, while the `ByteBuffer` natively supports serializing a `Double`, the format in which the value is serialized is not strongly specified, so the preference is to explicitly ensure a standard representation using `Double.doubleToRawLongBits()` and `Double.longBitsToDouble().`

clients/src/main/java/org/apache/kafka/common/utils/ByteUtils.java

```
    /**
     * Read a double-precision 64-bit format IEEE 754 value.
     *
     * @param buffer The buffer to read from
     * @return The long value read
     */
    public static double readDouble(ByteBuffer buffer) {
        return Double.longBitsToDouble(buffer.getLong());
    }

    /**
     * Write the given double following the double-precision 64-bit format IEEE 754 value into the buffer.
     *
     * @param value The value to write
     * @param buffer The buffer to write to
     */
    public static void writeDouble(double value, ByteBuffer buffer) {
        buffer.putLong(Double.doubleToRawLongBits(value));
    }
```

The protocol type definition:

clients/src/main/java/org/apache/kafka/common/protocol/types/Type.java

```
    public static final DocumentedType DOUBLE = new DocumentedType() {
        @Override
        public void write(ByteBuffer buffer, Object o) {
            ByteUtils.writeDouble((Double) o, buffer);
        }

        @Override
        public Object read(ByteBuffer buffer) {
            return ByteUtils.readDouble(buffer);
        }

        @Override
        public int sizeOf(Object o) {
            return 8;
        }

        @Override
        public String typeName() {
            return "DOUBLE";
        }

        @Override
        public Double validate(Object item) {
            if (item instanceof Double)
                return (Double) item;
            else
                throw new SchemaException(item + " is not a Double.");
        }

        @Override
        public String documentation() {
            return "Represents a double-precision 64-bit format IEEE 754 value. " +
                    "The values are encoded using eight bytes in network byte order (big-endian).";
        }
    };
```

In `generator/src/main/java/org/apache/kafka/message/MessageGenerator.java`, the following operations will be used (code omitted for brevity):

generator/src/main/java/org/apache/kafka/message/MessageGenerator.java

```
Hash code: Double.hashCode(value)

Empty value: (double) 0

Parsing a default value string: Double.parseDouble(defaultValue)
```

Compatibility, Deprecation, and Migration Plan

- No deprecation or migration necessary.
- Compatibility
    - Let's say existing message DescribeConfigsRequest added a `double` field.
        - If the client is old but server is new, then it won't attempt to set the double field, so it won't be serialized in the request.
        - If the client is new but server is old, then the server won't understand the request due to a version mismatch (this setup is generally not advised).
    - If a new request with a `double` field is added, both client and server must be aware of the message, and therefore will be aware of the `double` message type.

Rejected Alternatives

- Requiring the application to serialize into an existing RPC type (int64). This is prone to error and introduces additional complexity in areas where the application shouldn't have to worry about it.