KIP-937: Improve Message Timestamp Validation

Authors Mehari Beyene, Divij Vaidya

Status

Current state: *Accepted*

Discussion thread: *here*

Vote thread: here

JIRA: *KAFKA-14991*

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

Motivation

In Kafka, the introduction of the timestamp field in the message format through KIP-32 brought along two additional configurations: *log.message.timestamp.type* and *log.message.timestamp.difference.max.ms*.

By default, the configuration values are *log.message.timestamp.type=CreateTime* and *log.message.timestamp.difference.max.ms=9223372036854775807 (java.lang.Long.MAX_VALUE)*. This allows Producers to send messages with timestamps as far back as the minimum representable timestamp and ahead in the future at a scale of 100s of years. (**Side Note:** We could potentially change the default value of *log.message.timestamp.difference.max.ms* to something more sensible but that is not the motivation for this KIP.)

While there can be valid use cases for messages with past timestamps for the purpose of replaying old messages, messages with future timestamps are inherently inaccurate and can lead to unexpected log rotation behavior. Kafka users have encountered problems due to misconfigured producers, such as using nanoseconds instead of milliseconds for the message timestamp. This anecdotal evidence from a Medium article highlights the challenges associated with this issue.

The motivation behind this proposal is to improve the validation logic for message timestamps by rejecting messages with future timestamps and providing a descriptive exception. This will help improve data integrity and prevent potential pitfalls caused by inaccurate timestamp handling.

Public Interfaces

There will be no change to existing public interfaces. However, when a producer's record is rejected due to the timestamp validation, we will return error code 32 (INVALID_TIMESTAMP) with the error message "*Timestamp [record timestamp] of message with offset [record offset] is out of range. The timestamp should be within [[brokerTime - log.message.timestamp.before.max.ms], [brokerTime + log.message.timestamp.after.max.ms:]]."*

Note that the error message will remain the same as the current timestamp validation, except that we will be introducing two new configurations: `log.message.timestamp.before.max.ms` and `log.message.timestamp.after.max.ms`. Additionally, the configuration log.message.timestamp.difference.max.ms will be deprecated. Details of these configuration changes are discussed in the Proposed Changes section.

Proposed Changes

We will introduce two new configurations: *log.message.timestamp.before.max.ms* and *log.message.timestamp.after.max.ms*. Additionally, we will deprecate the existing configuration *log.message.timestamp.difference.max.ms*.

***log.message.timestamp.before.max.ms*:** This configuration defines that the message timestamp must always be earlier than or equal to the broker's timestamp, with a maximum allowable difference determined by the value of *log.message.timestamp.before.max.ms*.

- Type: long
- Default: 9223372036854775807
- Valid Values: [0,...]
- Update Mode: cluster-wide

Note that the default value of *log.message.timestamp.before.max.ms* is kept the same as *log.message.timestamp.difference.max.ms* to maintain backward compatibility.

log.message.timestamp.after.max.ms: This configuration defines that the message timestamp can not be later than the broker's timestamp, with a maximum allowable difference determined by the value of *log.message.timestamp.after.max.ms*.

- Type: long
- Default: 9223372036854775807
- Valid Values: [0,...]
- Update Mode: cluster-wide

Note that the default value of *log.message.timestamp.after.max.ms* is kept the same as log.message.timestamp.difference.max.ms to maintain backward compatibility and not introduce a breaking change for clients that are sending messages with future timestamps. However, in a future major version bump, we will change this default value to 3600000 milliseconds (one hour). To prepare users for this change in the future, we will add a WARN-level log when we encounter messages with future timestamps that are ahead of the proposed one-hour threshold.

Timestamp Validation Logic

The validation occurs before appending a message to the active segment of the local log. If *log.message.timestamp.type=LogAppendTime*, the server overwrites the timestamp with the broker's timestamp and proceeds without further validation related to the record timestamp. Note that variations exist when the message is compressed or not compressed, but the existing logic remains unchanged. Regardless of compression, the timestamp is always overwritten with the broker's current time.

We iterate through each batch included in the write request, and the validation logic in sudo code for each record looks like the following:

New Validation Logic

```
if message.timestamp <= broker.timestamp:
    time_difference = broker.timestamp - message.timestamp
    if time_difference <= log.message.timestamp.before.max.ms:
        # Validation passed for timestamp before
        # Proceed with further logic
    else:
        # Validation failed for timestamp before
        return Error code 32 (INVALID_TIMESTAMP)
else if message.timestamp > broker.timestamp:
    time_difference = message.timestamp - broker.timestamp
    if time_difference <= log.message.timestamp.after.max.ms:
        # Validation passed for timestamp after
        # Proceed with further logic
    else:
        # Validation failed for timestamp after
        return Error code 32 (INVALID_TIMESTAMP)
```

For comparison, the validation logic above replaces the current validation, which is represented by the following sudo code:

Existing Validation Logic

```
time_difference = absolute(message.timestamp - broker.timestamp)

if time_difference > log.message.timestamp.difference.max.ms:
    # Validation failed for timestamp difference
    return Error code 32 (INVALID_TIMESTAMP)
else:
    # Validation passed for timestamp difference
    # Proceed with further logic
```

It is important to note that if a validation fails, even for a single record within a batch, **we reject the entire batch**. This behavior remains consistent for both timestamp validation scenarios: records with future timestamps and records with past timestamps.

Please note that the description above omits details of other orthogonal validations such as key verification in compacted topics, offset contiguity verification, and batch record count verification.

Compatibility, Deprecation, and Migration Plan

- The configuration *log.message.timestamp.difference.max.ms* will be deprecated.
- The semantics of the configuration regarding time differences in the past and future were not explicitly defined with the configuration *log.message.timestamp.difference.max.ms*. With this KIP, message timestamps with past and future values are now validated using different default values, defined by the configurations *log.message.timestamp.before.max.ms* and *log.message.timestamp.after.max.ms* respectively.
- After this KIP messages with future timestamps are validated using the new configuration log.message.timestamp.after.max.ms. This will result in for some messages to be rejected that were previously accepted. We consider this an acceptable breaking change.

There are no other changes to public interfaces that will impact clients.

Test Plan

- This change will be tested via unit test.
- Additional verification will be done by recreating the issue manually and verifying that the correct error message is returned.

Future Work: Update Message format to include both Client Timestamp and LogAppend Timestamp

In the current message format (v2), a single Timestamp field is used, qualified by the timestamp type (0 for CreateTime, 1 for LogAppendTime). However, this design limits our selection to either the client timestamp or the broker's append time for log retention logic.

Nevertheless, both the client and broker timestamps are relevant as they capture different information and can be used to evaluate retention logic differently. The client timestamp closely represents the actual event timestamp as assigned by the producer. On the other hand, the broker's timestamp only indicates the time of log append, which can be considered an implementation detail.

To address this limitation, we have considered a modification that captures both the CreateTime and LogAppendTime, allowing for a more nuanced representation and validation of timestamps. However, the proposed change in this KIP does not involve updating the message format and incrementing the message version to v3. Such an update is a major undertaking that requires modifications across multiple components. We maintain a wiki entry that tracks the motivations for updating the message format, and it has been updated to include the proposal of capturing both timestamps.

Rejected Alternatives

Add a constant, TIME_DRIFT_TOLERANCE, with a one-hour value and use it to validate message timestamps with future values. Example code: https://github.com/apache/kafka/pull/13709

Pros:

- **Simplicity**: Hardcoding a constant value simplifies the validation logic by eliminating the need for additional configurations. It reduces complexity and potential configuration errors.

Cons:

- **Lack of Flexibility**: Users may have use cases that require extending the future timestamp validation beyond the default one hour, and the hardcoded constant value restricts this flexibility.
- **Compatibility Issues:** The hardcoded constant value of one hour can create compatibility issues when upgrading existing clusters or integrating with existing producers.