KIP-633: Deprecate 24-hour Default Grace Period for Windowed Operations in Streams

Status

Current state: *Accepted*

Discussion thread: *here*

JIRA:

Unable to render Jira issues macro, execution error.

Unable to render Jira issues macro, execution error.

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

Motivation

The grace period is a parameter of windowed operations such as Window or Session aggregates, or stream-stream joins. This config determines how long after a window ends any new data will still be processed. Records coming in after the grace period has elapsed will be dropped from those windows.

The current default value is 24hours, which has caused continuous problems and confusion for users of suppression since it means results won't show up for 24 hours. We'd like to drop this default value and give users a better way to make an informed decision in selecting a grace period.

Public Interfaces

All the classes extending from Windows will be aligned with the new approach to grace period. Two new static constructors will be added, one which accepts a grace period and one which indicates to use no grace period and close the window immediately when the window ends.

```
public class TimeWindows {

    @Deprecated
    public static TimeWindows of(Duration size);

    @Deprecated
    public TimeWindows grace(final Duration afterWindowEnd);

    // New
    public static TimeWindows ofSizeWithNoGrace(final Duration size);

    // New
    public static TimeWindows ofSizeAndGrace(final Duration size, final Duration afterWindowEnd);

}
```

```
public class SessionWindows {

    @Deprecated
    public static SessionWindows with(final Duration inactivityGap);

    @Deprecated
    public SessionWindows grace(final Duration afterWindowEnd);

    // New
    public static SessionWindows ofInactivityGapWithNoGrace(final Duration inactivityGap);

    // New
    public static SessionWindows ofInactivityGapAndGrace(final Duration inactivityGap, final Duration
afterWindowEnd);
}
```

```
public class JoinWindows {

    @Deprecated
    public static JoinWindows of(final Duration timeDifference);

    @Deprecated
    public JoinWindows grace(final Duration afterWindowEnd);

    // New
    public static JoinWindows ofTimeDifferenceWithNoGrace(final Duration timeDifference);

    // New
    public static JoinWindows ofTimeDifferenceAndGrace(final Duration timeDifference, final Duration
afterWindowEnd);
}
```

```
public class SlidingWindows {

    @Deprecated
    public static SlidingWindows withTimeDifferenceAndGrace(final Duration timeDifference, final Duration
afterWindowEnd);

    // New
    public static SlidingWindows ofTimeDifferenceWithNoGrace(final Duration timeDifference);

    // New
    public static SlidingWindows ofTimeDifferenceAndGrace(final Duration timeDifference, final Duration
afterWindowEnd);
}
```

Proposed Changes

The underlying principle here is that grace period is a fundamental concept in Kafka Streams, not an advanced feature which we only recommend for advanced users and which most applications can ignore. At the same time, sometimes a new user just wants to get a demo up and running without necessarily diving into every detail of a windowed operation. An ideal API is one in which the user is forced to at least acknowledge – for example by picking some auto-complete method in their IDE – that they are choosing one set of semantics over another.

Therefore, all existing APIs which don't accept a grace period and apply a default grace period of 24 hours will be removed. These will be replaced with two new APIs: one which ends in `WithNoGrace` and applies a grace period of 0, and one which takes the grace period as a required parameter. In constructing a windowed operation, users must choose between one of these two APIs and make a conscious decision whether to select a grace period or ignore that parameter for the time being.

To elaborate further, the current APIs will continue to have the grace period of 24 hours, however the new APIs will have the option of setting a grace period or having a default grace period of zero milliseconds if none is specified in the method signature. The *NoGrace() counterparts of the API methods will have 0 milliseconds as the grace period and the corresponding *AndGrace() API methods will have the ability to specify the grace period in the calling code.

Semantic Change

For stream-stream left/outer join, we will in addition apply a semantic improvement by enabling

Unable to render Jira issues macro, execution error.

if the new `JoinWindows.ofTimeDifferenceWithNoGrace` or `JoinWindows.ofTimeDifferenceAndGrace()` are used. If the existing `JoinsWindows.of()` is used, the old semantics to emit left/outer join results eagerly will be used. This will allow users to opt-into the new feature. The reason why we cannot apply KAFKA-10847 by default is the default grace period of 24h in the existing API (cf.

Unable to render Jira issues macro, execution error.

) that may lead to emitting left/outer join result very delayed and could break existing applications.

Compatibility, Deprecation, and Migration Plan

Users who currently rely on any of the deprecated methods will need to migrate to one of the two new APIs, and make a conscious decision to skip the grace period and close a window immediately, apply the old default of 24 hours, or choose a new grace period entirely.

Rejected Alternatives

1. No new or deprecated APIs, and just change the default to 0
    a. This was rejected because it poses a real risk to anyone using the current default who upgrades without carefully reading the upgrade guide and making the required changes. They would silently begin to drop out-of-order updates that they had previously processed without a problem, and the only hint that this was happening would be to carefully monitor and react upon the warnings in the logs or the dropped records metric.
    b. It also doesn't solve the underlying problem which has lead us to change the default grace period: that users can easily miss the concept of a grace period entirely. Many of them have been falling back on the 24hr default not because they specifically decided this was appropriate for their use case, but because they simply didn't notice. Swapping out the 24hr default with 0 would just mean that instead of getting questions from suppression users about why they don't see any results, we would have *any* user (ie, not just those with suppression) be impacted and fail to handle out-of-order data, something that Kafka Streams advertises we do out-of-the-box
2. Only add one new API, which requires choosing a grace period (ie, no default at all)
    a. This was rejected because many feel that the grace period is a more advanced concept and can be a stumbling block for new users just trying out Kafka Streams for the first time
3. Just throw an IllegalArgumentException on runtime if the grace period is not set
    a. The advantage is that we don't need to change any APIs, and remain with the short method names
    b. The disadvantage here is that it's a slower feedback loop for users, and it may be frustrating for them to start up an app that compiles only for it to crash because they failed to specify a parameter that the interface did not seem to require.
    c. Also, this will crash the app of any default grace period user who upgrades without changing their code, which seems likely to be a very significant subset of users
4. Use the incremental builder pattern in the new APIs
    a. This is closest to the proposal in this KIP, being an alternative to the specific new APIs we introduce. Personally I think this is a fine alternative, but ultimately chose to go another direction for the reasons listed below
    b. The advantage here is that individual method names are shorter, and each method only needs to require one parameter so the javadocs don't need to repeat the same parameters across multiple methods (eg `windowSize`)
    c. The disadvantage is that it's a larger overhaul to switch over to the builder pattern, and it's inconsistent with the way we do things elsewhere in Streams so users might have a difficult time adjusting or figuring out why their code won't compile