KIP-645: Replace Windows with a proper interface

Status

Current state: *Stalled*

Discussion thread: -

JIRA: Unable to render Jira issues macro, execution error.

POC: https://github.com/apache/kafka/pull/9031

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

Motivation

Presently, windowed aggregations in KafkaStreams fall into two categories:

- Windows
  - TimeWindows
  - UnlimitedWindows
  - JoinWindows
- SessionWindows

Unfortunately, Windows is an abstract class instead of an interface, and it forces some fields onto its implementations.

This has led to a number of problems over the years, but so far we have been able to live with them.

Extension points in an API should almost always be interfaces, since it allows the system to define just the basics of what it needs to know, and implementers can fill in that information any way they see fit. A lot of extra baggage comes with an abstract class, though. It forces constructors and fields upon implementers, which may not be appropriate for the semantics of the implementation. In the case of a multi-level hierarchy, it's not possible to know which layer is responsible for maintaining the field, which leads to bugs and semantic ambiguity.

All of these problems came to the surface when I implemented grace period and needed to move retention time and segments into the store definition. It should have been as simple as deprecating a couple of methods, but instead there were constructors and fields to contend with, and computing the actual retention time to use became enormously complex. For example, should we resolve the value returned by the getter method or the one in the abstract class's public field? I did the best I could, but the resulting code is very difficult to understand. My plan at the time was to deprecate everything except the necessary getter methods, hoping to convert the abstract class into "effectively" an interface, although it would never be possible to actually convert it into an interface, and there would be no way to ensure a simple design bug wouldn't return us to the same unfortunate state in the future. Plus, we just have this perpetual bizarre state in which we want Windows to be exactly like an interface, except that it's an abstract class. Better to just fix it.

My motivation to bring this up now is that I have had several recent conversations in which people were considering new extensions of Windows. We considered extending Windows in KIP-450, and there have been several requests to add calendar-aligned windows or other kinds of windows.

I think the approach I'm proposing here is safer than my earlier plan: Instead of living with an interface-like abstract class Windows, I'm proposing to slice it out of the hierarchy. We add a new interface on top of it, and migrate the DSL to expect that interface. We deprecate Windows itself, causing custom stores to stop extending Windows and just implement the interface instead. Then, once we remove Windows in 3.0 or 4.0, the API is safe and clean.

At the same time, we can make a small adjustment to the interface to correct a design bug that prohibits calendar-aligned windows like daily or monthly windows. The TimeWindows algorithm doesn't require fixed-size windows, just enumerable ones, given a record timestamp. I have already corrected the algorithm in the POC PR, and the only remaining use case for the concept of "window size" is to provide a lower bound for retention time. Thus, I'm proposing to replace Windows#size() with EnumerableWindowDefinition#maxSize(), so that variable-sized window definitions like "monthly windows" could give an upper bound like "32 days" on their size, ensuring windows would never get dropped from the store before they are closed.

Public Interfaces

1. Add new interface to take the place of Windows: EnumerableWindowDefinition
2. Add "implements EnumerableWindowDefinition" to TimeWindows and UnlimitedWindows
   a. Do not add the new interface to JoinWindows, which should not be part of this hierarchy. It will naturally become disconnected when we remove Windows
3. Add "implements EnumerableWindowDefinition" to Windows and deprecate it.
4. Swap out the argument type in both windowBy methods

New interface to take the place of Windows: EnumerableWindowDefinition

FixedSizeWindowDefinition

```
/**
 * The window specification for windows that can be enumerated for a single event based on its time.
 * <p>
 * Grace period defines how long to wait on out-of-order events. That is, windows will continue to accept new
records until {@code stream_time >= window_end + grace_period}.
 * Records that arrive after the grace period passed are considered <em>late</em> and will not be processed but
are dropped.
 * <p>
 * Window state is stored until it is no longer needed, measured from the beginning of the window. To assist
the store in not discarding window state too soon, this definition
 * also includes a {@code maxSize()}, indicating the longest span windows can have between their start and end.
This does not need to include the grace period.
 *
 * @param <W> type of the window instance
 */
public interface EnumerableWindowDefinition<W extends Window> {


    /**
     * List all possible windows that contain the provided timestamp,
     * indexed by non-negative window start timestamps.
     *
     * @param timestamp the timestamp window should get created for
     * @return a map of {@code windowStartTimestamp -> Window} entries
     */
    Map<Long, W> windowsFor(final long timestamp);

    /**
     * Return an upper bound on the size of windows in milliseconds.
     * Used to determine the lower bound on store retention time.
     *
     * @return the maximum size of the specified windows
     */
    long maxSize();

    /**
     * Return the window grace period (the time to admit
     * out-of-order events after the end of the window.)
     *
     * Delay is defined as (stream_time - record_timestamp).
     */
    long gracePeriodMs();
}
```

Add EnumerableWindowDefinition to TimeWindows and UnlimitedWindows

new implementations

```
@SuppressWarnings("deprecation") // Remove this suppression when Windows is removed
public final class TimeWindows extends Windows<TimeWindow> implements EnumerableWindowDefinition<TimeWindow> {
// ... nothing else needs to change
}

@SuppressWarnings("deprecation") // Remove this suppression when Windows is removed
public final class UnlimitedWindows extends Windows<UnlimitedWindow> implements
EnumerableWindowDefinition<UnlimitedWindow> {
// ... nothing else needs to change
}
```

Add EnumerableWindowDefinition to Windows and deprecate it

Includes deprecating size() and delegating maxSize() to size() to for compatibility.

deprecation

```
/**
 * ... the javadoc doesn't change
 *
 * @deprecated since 2.7 Implement EnumerableWindowDefinition instead.
 */
@Deprecated
public abstract class Windows<W extends Window> implements EnumerableWindowDefinition<W> {
    /**
     * Return an upper bound on the size of the specified windows in milliseconds.
     * Used to determine the lower bound on store retention time.
     *
     * @return the size of the specified window
     */
    public long maxSize() {
        return size();
    }

    /**
     * Return the size of the specified windows in milliseconds.
     *
     * @return the size of the specified windows
     * @deprecated since 2.7 Override maxSize() instead.
     */
    @Deprecated
    public abstract long size();
}
```

Swap out the argument type in windowBy

Note, because Windows now implements EnumerableWindowDefinition, all existing implementations of Windows will automatically work with this change, so there is no compatibility concern.

```
public interface CogroupedKStream<K, VOut> {
...
-    <W extends Window> TimeWindowedCogroupedKStream<K, VOut> windowedBy(final Windows<W> windows);

+    <W extends Window> TimeWindowedCogroupedKStream<K, VOut> windowedBy(final EnumerableWindowDefinition<W>
windows);
...

}
```

```
public interface KGroupedStream<K, V> {
...
-    <W extends Window> TimeWindowedKStream<K, V> windowedBy(final Windows<W> windows);
+    <W extends Window> TimeWindowedKStream<K, V> windowedBy(final EnumerableWindowDefinition<W> windows);
...
}
```

Compatibility, Deprecation, and Migration Plan

Windows is deprecated. Otherwise, no compatibility issues arise. We can remove Windows cleanly later on.

Rejected Alternatives

None (yet)