KIP-825: introduce a new API to control when aggregated results are produced

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

Currently in Kafka Streams, windowed aggregations always emit an aggregated result for each input record. The main problem with this approach is that users sometimes just need a final result when a window closes. For example, if a user defines a 1 day tumbling window

but before 1 day ends, there could be multiple results using existing output mechanism. This not only increases downstream system load, but also make it hard to interpret the results since most of the results are partial and the user can't tell which result is final. An existing solution

is to use suppressed.untilWindowCloses operator to buffer the upstream aggregated results and only output final results when windows close. There are several issues using existing suppress operators:

- It's semantically undesirable since outputting final results when window closes should be controlled by window level api.
- Existing suppress operator uses a separate in-memory buffer to buffer all results as well as a new changelog to support durability.
- The information suppress operator needs to output final result is also maintained in windowed operator. The suppress operator seems redundant which adds performance overhead of memory, CPU, disk and network. In addition, maintaining a new changelog topic also adds more operational overhead.

To resolve these issues, we propose to add a new API to control windowed aggregation output behavior and possibly other behavior later on.

Public Interfaces

In existing `TimeWindowedKStream` and `SessionWindowedKStream` interfaces.

```
public interface TimeWindowedKStream<K, V> {
    TimeWindowedKStream<K, V> emitStrategy(EmitStrategy strategy);
}

public interface SessionWindowedKStream<K, V> {
    SessionWindowedKStream<K, V> emitStrategy(EmitStrategy strategy);
}

public interface EmitStrategy {
    enum StrategyType {
        ON_WINDOW_CLOSE, // output final result
        ON_WINDOW_UPDATE;  // output for every record
    }

        StrategyType type();

    static EmitStrategy onWindowClose() {
        return new WindowCloseStrategy();
    }

    static EmitStrategy onWindowUpdate() {
        return new WindowUpdateStrategy();
    }
}

public class WindowCloseStrategy implements EmitStrategy {
        WindowCloseStrategy() {}

    StrategyType type() {
        return ON_WINDOW_CLOSE;
    }
}
```

I also propose to add following two metrics measuring the latency to emit final and number of records emitted for emit final:

Each exposed metric will have the following tags:

- thread-id = [thread ID],
- task-id = [task ID]
- processor-node-id = [node ID]

The following metrics will be exposed in the Kafka Streams' metrics

- emit-final-latency-max (max latency to emit final records when it COULD be emitted)
- emit-final-latency-avg (avg latency to emit final records when it could be emitted)
- emit-final-records-rate (rate of records emitted when it COULD be emitted)
- emit-final-records-total (total number of records emitted)

The recording level for all metrics will be DEBUG and their group will be stream-processor-node-metrics.

As for the implementation to support emit-final, for time windows (hopping, tumbling, sliding) we already have the needed API to range-over the finalized and immutable windows to emit them, however for session windows we do not yet have the corresponding APIs to do so. Hence as part of this KIP, we also propose to add a new range API in the SessionStore to support this feature. Note this will be only add on the SessionStore API not the ReadOnlySessionStore API so that it would not be exposed for interactive query usage.

```
public interface SessionStore<K, AGG> extends StateStore, ReadOnlySessionStore<K, AGG> {

    /**
     * Return all the session window entries that ends between the specified range (both ends are inclusive).
     * This function would be used to retrieve all closed and immutable windows.
     *
     * @param earliestSessionEndTime earliest session end time to search from, inclusive
     * @param latestSessionEndTime latest session end time to search to, inclusive
     */
    default KeyValueIterator<Windowed<K>, AGG> findSessions(final long earliestSessionEndTime,
                                                            final long latestSessionEndTime) {
        throw new UnsupportedOperationException(
                "This API is not supported by this implementation of SessionStore.");
    }

    // other existing functions
}
```

Proposed Changes

We introduce several options of API changes discussed above to support output final result for windowed aggregations. We also introduce two metrics to measure the emit final latency as well as the number of records emitted.

Compatibility, Deprecation, and Migration Plan

We also plan to introduce a new default state store supporting more efficient time range lookup for emit final windows. This won't be backward compatible with existing state store and windowed aggregation type. However,

if users continue to use their existing state stores, this should be backward compatible with worse performance maybe.

Rejected Alternatives

First rejected option

```
public interface KGroupedStream<K, V> {
    <W extends Window> TimeWindowedKStream<K, V> windowedBy(final Windows<W> windows); // Already exist
    TimeWindowedKStream<K, V> windowedBy(final SlidingWindows windows);    // Already exist
    SessionWindowedKStream<K, V> windowedBy(final SessionWindows windows); // Already exist

    <W extends Window> TimeWindowedKStream<K, V> windowedBy(final Windows<W> windows, EmitConfig config); // new
    TimeWindowedKStream<K, V> windowedBy(final SlidingWindows windows, EmitConfig config);    // New
    SessionWindowedKStream<K, V> windowedBy(final SessionWindows windows, EmitConfig config); // New
}

public interface EmitConfig {
    enum ConfigType {
        EMIT_FINAL, // output final result
        EMIT_EAGER;  // output for every record
    }

    ConfigType type();

    static EmitConfig emitFinal() {
        return new EmitFinalConfig();
    }

    static EmitConfig emitEager() {
        return new EmitEagerConfig(); // EmitEagerConfig will be similar to EmigFinalConfig
    }
}

public class EmitFinalConfig implements EmitConfig {
    EmitFinalConfig() {}

    ConfigType type() {
        return EMIT_FINAL;
    }
}
```

This option is rejected because

1. It creates more overloading of the `windowdedBy` function.
2. It operates on `KGroupedStream` and tries to make it a windowed stream as well as configuring the emit policy for the stream. I think this is against the builder pattern.

Second rejected option

In existing `Windows` , `SlidingWindows` and `SessionWindows` classes.

```
public abstract class Windows<W extends Window> {
    public abstract EmitConfig getEmitConfig(); // New
}

public final class SlidingWindows {
     public abstract EmitConfig getEmitConfig(); // New
}

public final class SessionWindows {
     public abstract EmitConfig getEmitConfig(); // New
}

public interface EmitConfig {
    enum ConfigType {
        EMIT_FINAL, // output final result
        EMIT_EAGER;  // output for every record
    }

    ConfigType type();

    static EmitConfig emitFinal() {
        return new EmitFinalConfig();
    }

    static EmitConfig emitEager() {
        return new EmitEagerConfig(); // EmitEagerConfig will be similar to EmigFinalConfig
    }
}

public class EmitFinalConfig implements EmitConfig {
    EmitFinalConfig() {}

    ConfigType type() {
        return EMIT_FINAL;
    }
}
```

This option is rejected because

1. Emit config should operator on the stream and configure how it should be outputted. Window definition shouldn't control that.