**Window Functions**

**Definition**

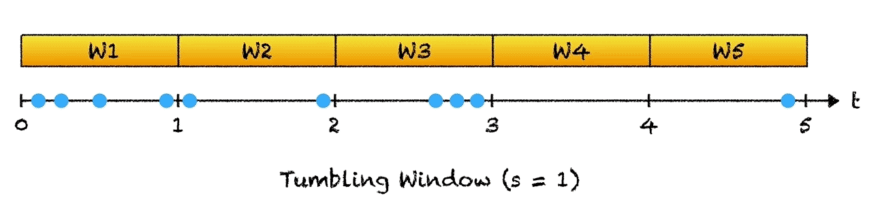
A window function assigns events in your stream to windows. To be precise it is more a window relation rather than a function because it theoretically does not have to assign all events to windows, i.e. it is not total, and it can assign an event to multiple windows. In addition it is neither surjective (not all windows have to contain events) nor injective (a window can contain multiple events). Nevertheless we are going to stick to the mathematically incorrect term window function.

Given a window function and a stream of data we can compute aggregates on events inside each window. As mentioned earlier an event might be assigned to multiple windows or a window might have no events assigned at all, depending on the selected window function. This is important to keep in mind when working with the derived stream of aggregates as. For example a graph of event counts based on overlapping windows will look very different from a graph based on counts computed from distinct windows.

Martin Kleppmann mentions four commonly used window functions [1]: *Tumbling window*, *hopping window*, *sliding window* and *session window*. The next sections are going to explain each of them in detail.

**Tumbling Window**

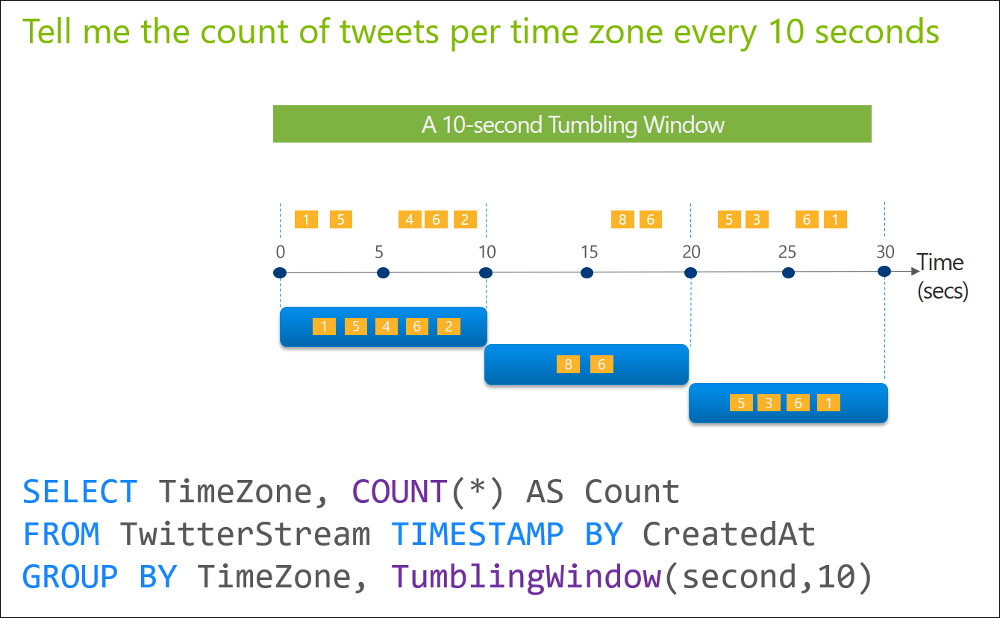
A tumbling window has a fixed length. The next window is placed right after the end of the previous one on the time axis. Tumbling windows do not overlap and span the whole time domain, i.e. each event is assigned to exactly one window. You can implement tumbling windows by rounding down the event time to the nearest window start. The following animation illustrates a tumbling window of length 1.

[](https://res.cloudinary.com/practicaldev/image/fetch/s--yuo1M_Sc--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_66%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/c1n1b45h73yahej0cx5k.gif)

Because tumbling windows are only configured through a single property, the window length *s*, and they include every event exactly once, they are often used for simple reporting. You can use tumbling windows to sum all incoming requests towards your server within a 1 minute window and then display a graph where each minute corresponds to one data point.

**Azure Stream Analytics:**

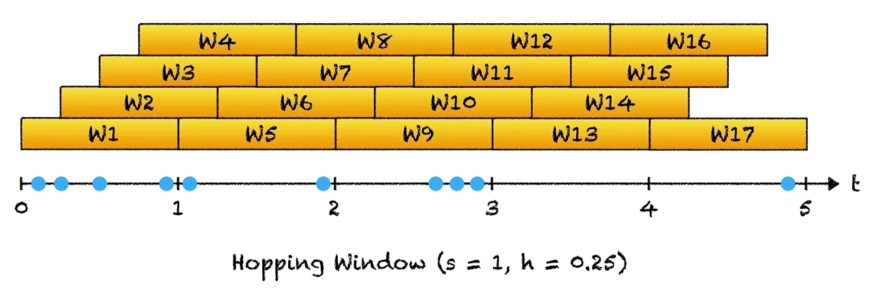
Tumbling windows are a series of fixed-sized, non-overlapping and contiguous time intervals. The following diagram illustrates a stream with a series of events and how they are mapped into 10-second tumbling windows.



**Hopping Window**

Like tumbling windows, hopping windows also have a fixed length. However they introduce a second configuration parameter: The hop size *h*. Instead of moving the window of length *s* forward in time by *s* we move it by *h*.

This means that tumbling windows are a special case of hopping windows where *s = h*. If *s > h* windows are overlapping and if *s < h* some events might not be assigned to any window. The following animation illustrates a hopping window of length 1 with hop size 0.25. It’s common to choose *h* to be fraction of *s*.

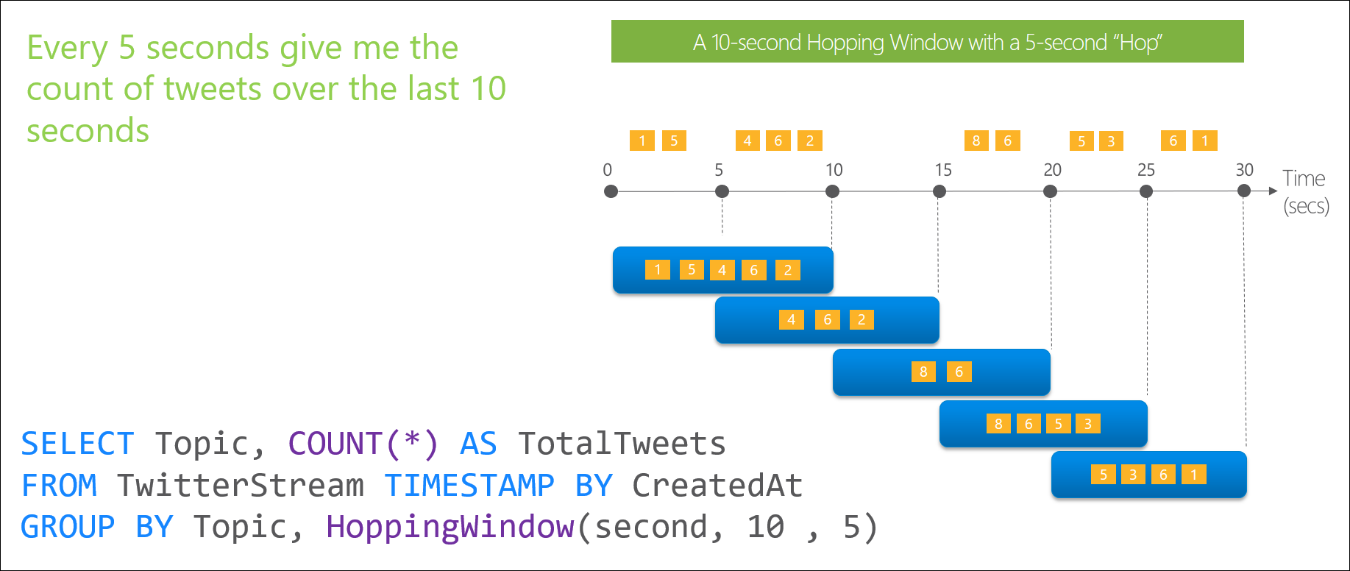
[](https://res.cloudinary.com/practicaldev/image/fetch/s--96wcSgjP--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_66%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/13sikpcme4mhsliibhe6.gif)

Hopping windows where *h* is a fraction of *s* can be implemented by computing tumbling windows of size *h* and aggregating them into a bigger hopping window. A common use case for hopping windows are moving average computations.

**Azure Stream Analytics**

Unlike tumbling windows, hopping windows model scheduled overlapping windows. A hopping window specification consist of three parameters: the timeunit, the windowsize (how long each window lasts) and the hopsize (by how much each window moves forward relative to the previous one). Additionally, offsetsize may be used as an optional fourth parameter. Note that a tumbling window is simply a hopping window whose ‘hop’ is equal to its ‘size’.

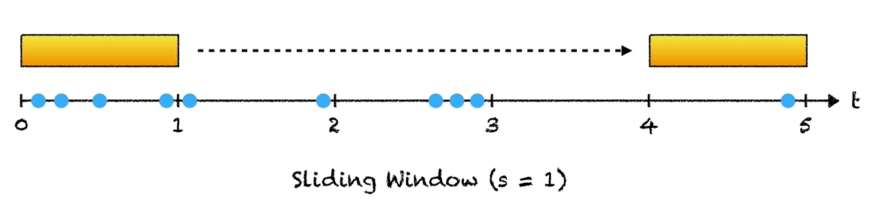
The following illustration shows a stream with a series of events. Each box represents a hopping window and the events that are counted as part of that window, assuming that the ‘hop’ is 5, and the ‘size’ is 10.

****

**Sliding Window**

Sliding windows can be viewed as hopping windows with *h → 0*. While they are discretizing the input stream the derived aggregated stream is not discrete. A sliding window moves along the time axis, grouping together events that happen within the window length *s*.

However, as our data points are discrete, we can implement a sliding window by moving forward based on actual events rather than continuously in time. A new window is created whenever an event enters or exits the length of the sliding window moving forward. This mathematically corresponds to a deduplication of all possible windows based on the set of events that have been assigned to them. The following figure illustrates a sliding window of length 1.

[](https://res.cloudinary.com/practicaldev/image/fetch/s--iQqSZzGW--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_66%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/i6gei09zof2p0vkmruje.gif)

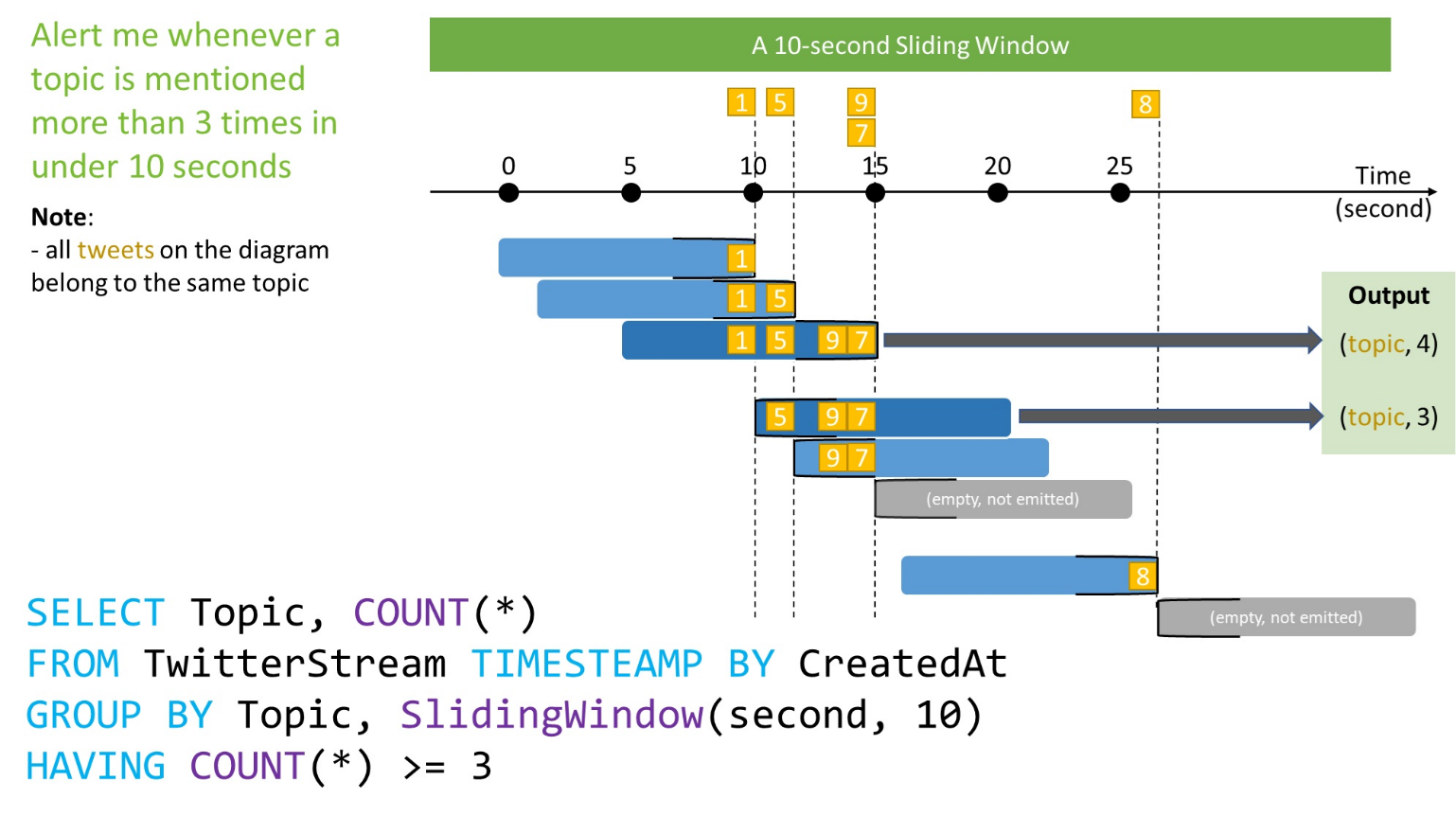
Sliding windows are, for example, used to compute moving averages. What makes them unique is that they provide a resolution based the event time pattern in your stream rather than a fixed one. If events are denser you will get a higher resolution of your moving aggregate. If no events are coming in, the aggregate stream stays the same without emitting new values.

Note that sliding windows are not always implemented the same way. In some tools the aggregation computation is only triggered when a new event *enters* the window but not if an old event *exits*. Make sure to check the documentation or source code of the tool you are using.

**Azure Stream Analytics**

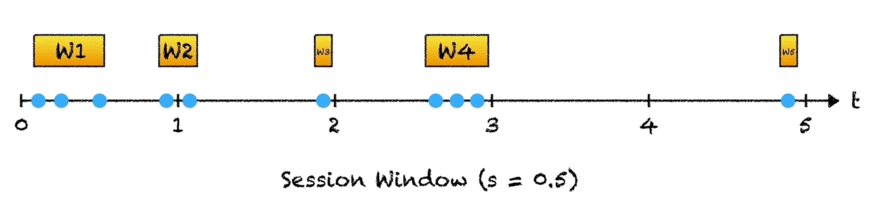
When using a sliding window, the system is asked to logically consider all possible windows of a given length. As the number of such windows would be infinite, Azure Stream Analytics instead outputs events only for those points in time when the content of the window actually changes, in other words when an event entered or exits the window.

The following diagram illustrates a stream with a series of events and how they are mapped into sliding windows of 10 seconds.

****

**Session Window**

In contrast to the previous window functions session windows have a variable length. When using a session window function you need to specify a time threshold between consecutive events that must not be exceeded. The window will keep expanding as long as new events are coming in that are close enough in time. The animation below illustrates a session window with a threshold of 0.5.

[](https://res.cloudinary.com/practicaldev/image/fetch/s--CZ3fQBFT--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_66%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/2s7fvhumgbarpmtj5i2g.gif)

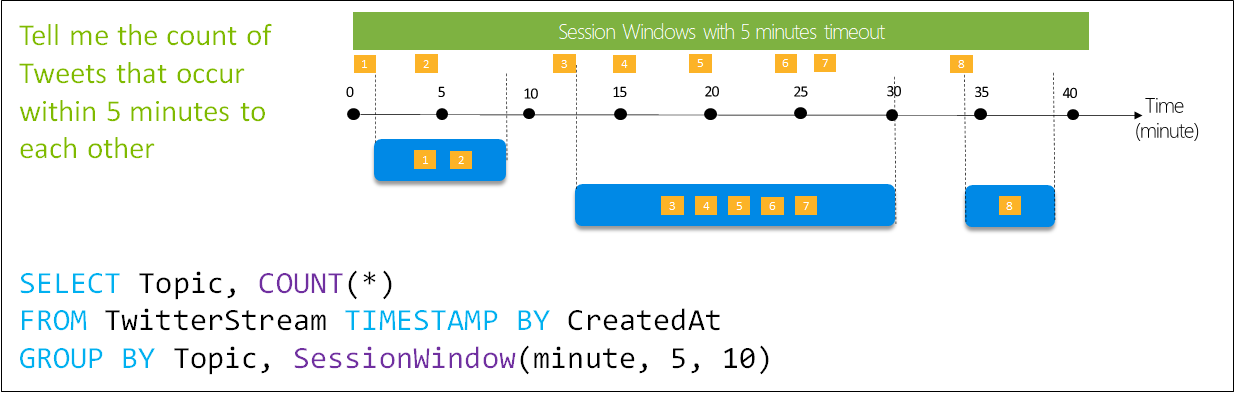
You can implement a session window by keeping the current events in a buffer and adding new events as long as they are within the specified session interval. As streams are unbounded sessions can theoretically grow indefinitely. Thus some implementations take a second parameter which represents the maximum session time or the maximum amount of events per session.

Session windows are useful to group together events that are expected to be related when they happen in close succession. The name suggests the prominent use case for this window function: Grouping clicks inside user sessions on your website. As long as the user keeps clicking within a short period of time your window function will aggregate all clicks in one session.

**Azure Stream Analytics**

Session windows group events that arrive at similar times, filtering out periods of time where there is no data. Session window function has three main parameters: timeout, maximum duration, and partitioning key (optional).

The following diagram illustrates a stream with a series of events and how they are mapped into session windows of 5 minutes timeout, and maximum duration of 10 minutes.



A session window begins when the first event occurs. If another event occurs within the specified timeout from the last ingested event, then the window extends to include the new event. Otherwise, if no events occur within the timeout, then the window is closed at the timeout.

If events keep occurring within the specified timeout, the session window will keep extending until maximum duration is reached. Please note that the maximum duration checking intervals are set to be the same size as the specified max duration. For example, if the max duration is 10, then the checks on if the window exceed maximum duration will happen at t = 0, 10, 20, 30, etc. That means that the actual duration of a session window could then be up to twice maxDuration.

Thus mathematically, our session window ends if the following condition is satisfied:



When a partition key is provided, the events are grouped together by the key and session window is applied to each group independently. This is useful for cases where you need different session windows for different users or devices.