Event Windowing is a type of complex event processing pattern that is used to detect the occurrence of certain events within a specified time window

The pattern is typically used in scenarios like -

1. Aggregated Message Processing - Accumulating messages in a cache and only processing them when a specific message arrives in the event window
2. Fraud Detection - When certain types or occurrences of messages are detected in a certain time window. Ie Multiple credit card transaction received within a 5 min timeframe from a location that may not be the cardholders primary location

While Mulesoft does not provide CEP processor types OOB, ObjectStore is a very powerful mechanism to implement such patterns with high performance profiles

**Use Case 1**

UC1 illustrates a pattern wherein, if 5 messages are received in a 10 second window the detectwindowevent API will trigger an event. The trigger can be anything from sending out a notification message to a queue/email etc to creating a new payload or rejecting all cached transactions

The UC consists of an http listener with a specific endpoint. The idea is for the calling application to trigger this api each time an event is being processed. The implementation uses timestamps as keys and sets the payload of the event as a value.

The flow adds the event to the cache and subsequently uses object store operations to determine if the window criteria has been satisfied. Ordering the keys is important to figure out the first and last transaction timestamps. Since we are using the timestamp itself as the key, there is no need to enhance the payload with additional information.

Upon completion of the detection process, the flow simply removes the first transaction from the cache. This is an example of a self-maintaining FIFO model. There is no need for an extra process to manage the cache

**Use Case 2**

Very similar to UC1, except in this case we need to check for occurrences of a certain type of messages in a 2 sec window. The algorithm checks to see if message type A and B and received in a 2 second window. The logic is limited and will not work if messages are received faster than 2 sec intervals. The self-maintaining functionality would remove messages before they can be used in the candidate set for event detection

In this case it would have been beneficial to have a third process that would check the object store asynchronously, so we don't lose messages before they are properly processed within a specific window

It also includes an endpoint to clear the cache during testing process. This helped me when I was repeatedly tweaking the algorithm