

Working with Streaming Data

Till now we have worked with batch processing. Example would be – Webserver logs are copied to the Data Lake over the data and then a batch process would run at night to process the data and send it to an Analytical store for daily reporting purposes.

Real Time Processing – Here streams are captured in real-time and processed with minimal latency to generate real-time reports. Here processing of data needs to be done as fast as possible so that it does not block the incoming stream of data. All platforms should be available to ingest large amounts of data at a fast rate.

For real time message ingestions – Apache Kafka and Azure Events Hubs is used

For data storage – Azure Data Lake and Blob Storage

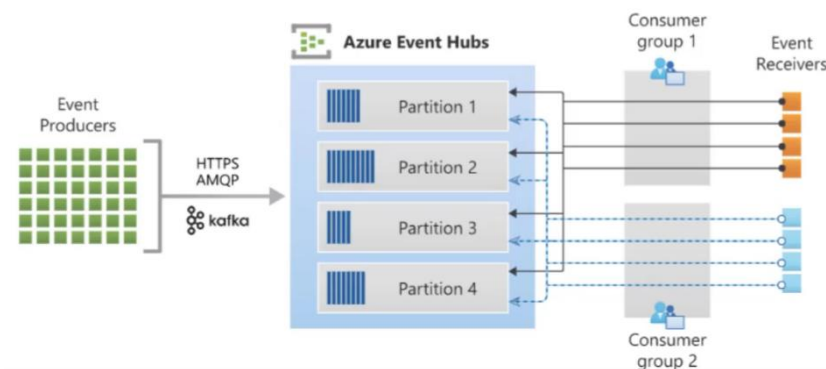
For Stream processing – Azure Stream Analytics and Spark Streaming

For Analytical Data Storage – Azure synapse

Azure Event Hubs

This is a big data streaming platform. Can receive and process millions of events per second. Can stream log data, telemetry data, any sort of event.

Events hub architecture –



In azure we would first create an Azure you'd create something called Events hubs namespace. As part of the namespace, we can create the Events hub. Then you can have sources that produce event and send it onto Azure Event hubs. Towards the end of the hub we can have a receiver that is taking in all the data that is being sent to the Azure events hub. Event receivers can take the data and process the data accordingly.

In Azure events hubs there are multiple partitions, which would help you ingest more data at a given time. Also helps event receivers take in data at a faster rate.

The different components –

- Event producers – an entity that sends data to an event hub. Events can be published using protocols like https, Apache kafka.
- Partitions – data is split across multiple partitions, allowing for better throughput of data onto the Events hub.
- Consumer groups – this is a view of the entire event hubs
- Event receivers – the entity that read the event data

Now we would be looking at an instance where a sql database would send metrics information onto events hub and then our receiver would be azure stream analytics.

Creating an instance of Azure Event Hub

Go to all resources > Create > Event Hubs > Create

On the create namespace screen select your subscription, your resource group, unique namespace, location, and pricing tier.

Here is the pricing page for the events hub for East US –

	Basic	Standard	Premium***	Dedicated*
Throughput unit (1 MB/s ingress, 2 MB/s egress)	\$0.015/hour	\$0.03/hour	Billed per Premium Unit (PU)	Billed per Capacity Unit (CU)
Ingress events	\$0.028 per million events	\$0.028 per million events	Included	Included
Capture		\$0.10/hour	Included	Included
Apache Kafka		✓	✓	✓
Schema Registry		✓	✓	✓
Max Retention Period	1 day	7 days	90 days	90 days
Storage Retention	84 GB	84 GB	1 TB per PU	10 TB per CU
Extended Retention**			\$0.10/GB/month (1 TB included per PU)	\$0.10/GB/month (10 TB included per CU)

1 throughput using allows us to process 1 MB/s as input and 2 MB/s as output. So, the basic tier has \$0.015/hour for using one throughput unit. The basic tier has a retention period of 1 day.

I choose the standard tier with 2 throughput units. Review and create. When the resource is created, go to the resource, click on the the “ + Event Hub “ button to create a new Event Hub. Give a name, partition count as 1, message retention at 7.

When we’re using the standard tier we can also use the capture feature. This will give you the ability to take the event and put it into storage accounts. We keep it off and create the event hub.

Once it is created, it would show up on the left under Entities in event hubs.

Sending and receiving events

For this part I am using a simple python code to send and receive the events using Event Hubs package available for python.

To follow along on your machine, install the required packages using the command –

```
pip install azure-eventhub
pip install azure-eventhub-checkpointstoreblob-aio
```

Next, we get the connection string to the event hubs from the Azure portal.

Go to the event hub that you just created. On the left you would see Shared Access policies.

Add one with the send permission. Click on it and copy the connection string primary key and paste it in the conn_str string in this code. Also give the name of your event hub in the second field –

```
import asyncio
from azure.eventhub.aio import EventHubProducerClient
from azure.eventhub import EventData

async def run():
    # Create a producer client to send messages to the event hub.
    # Specify a connection string to your event hubs namespace and
    # the event hub name.
    producer = EventHubProducerClient.from_connection_string(conn_str="EVENT HUBS NAMESPACE - CONNECTION STRING",
eventhub_name="EVENT HUB NAME")
```

```

async with producer:
    # Create a batch.
    event_data_batch = await producer.create_batch()

    # Add events to the batch.
    event_data_batch.add(EventData('First event '))
    event_data_batch.add(EventData('Second event'))
    event_data_batch.add(EventData('Third event'))

    # Send the batch of events to the event hub.
    await producer.send_batch(event_data_batch)

loop = asyncio.get_event_loop()
loop.run_until_complete(run())

```

Now create a container in your data lake gen2 account and get the connection string to the storage account to use it in the following code. Also create receive connection string on your event hub and use it here along with the event hub name just like in the previous code –

```

import asyncio
from azure.eventhub.aio import EventHubConsumerClient
from azure.eventhub.extensions.checkpointstoreblobaio import BlobCheckpointStore

async def on_event(partition_context, event):
    # Print the event data.
    print("Received the event: \"{}\" from the partition with ID: \"{}\"".format(event.body_as_str(encoding='UTF-8'), partition_context.partition_id))

    # Update the checkpoint so that the program doesn't read the events
    # that it has already read when you run it next time.
    await partition_context.update_checkpoint(event)

async def main():
    # Create an Azure blob checkpoint store to store the checkpoints.
    checkpoint_store = BlobCheckpointStore.from_connection_string("STORAGE ACCOUNT CONNECTION STRING", "CONTAINER NAME")

    # Create a consumer client for the event hub.
    client = EventHubConsumerClient.from_connection_string("EVENT HUB CONNECTION STRING",
consumer_group="$Default", eventhub_name="EVENT HUB NAME", checkpoint_store=checkpoint_store)
    async with client:
        # Call the receive method. Read from the beginning of the partition (starting_position: "-1")
        await client.receive(on_event=on_event, starting_position="-1")

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    # Run the main method.
    loop.run_until_complete(main())

```

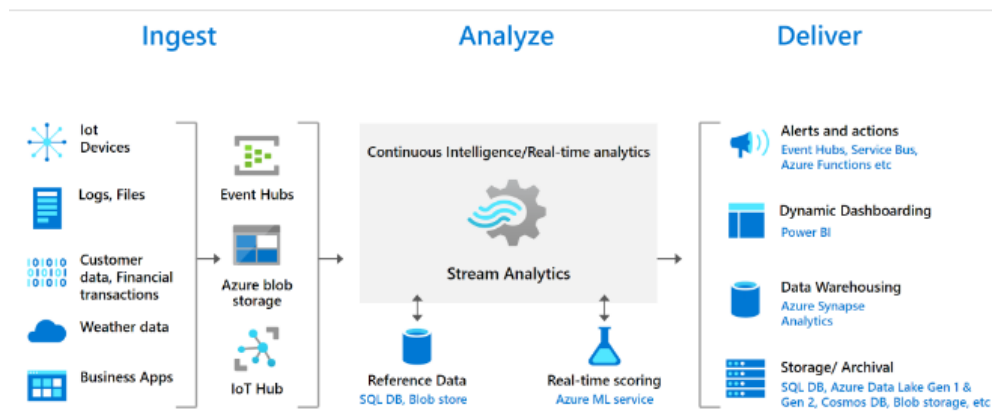
Once the first file is run and then the second file is run, we can see the following messages. There's being displayed twice because the first file was run twice by me –

```
'c:\Users\rayar\.vscode\extensions\ms-python.python-2021.10.1365161279\pythonFile
Hubs send and receive events\receive.py'
Received the event: "First event " from the partition with ID: "0"
Received the event: "Second event" from the partition with ID: "0"
Received the event: "Third event" from the partition with ID: "0"
Received the event: "First event " from the partition with ID: "0"
Received the event: "Second event" from the partition with ID: "0"
Received the event: "Third event" from the partition with ID: "0"
```

The Data lake container would also get some files and folders that would contain information related to the above event. This was a very simple example.

Azure Stream Analytics

A real time analytics and event processing service. Architecture looks like –



We can ingest data from a variety of different sources (IOT devices, log files, etc.). We can use services like event hubs to ingest that data or directly store that data onto Azure blob storage. We can then analyze this data in real time and then transfer that data onto a destination data store like a data warehouse or on a dynamic dashboard like PowerBI.

Creating a Stream Analytics job

On all resources, create a new resource > search Stream Analytics job > Create > Give a job name, select the subscription, resource group, location and streaming units as 3. Streaming units is representative of the amount of compute resources that would be given. Create.

Here in the inputs section which is selected from the left hand menu, we can add stream inputs from event hub, blob storage or the iot hub. Then there are outputs, which can be a variety of options, like the event hub, sql database, blob storage, PowerBI, etc. Once the inputs and outputs have been defined, then we would define our query which is like a sql based query.

Defining a Stream Analytics job

We would be making use of [this](#) visual studio project. We would be Sending data using this to the event hub and using stream analytics we would read that data and store it onto our dedicated sql pool.

Begin by going to you stream analytics and define your input.

- Add an Event hub input, give a name, select your namespace, event hub. It would also create a new event hub consumer group. In the authentication mode, select Connection string. For the rest leave as it is.
- Now we create a table in the dedicated sql pool to create a new table with the following query –

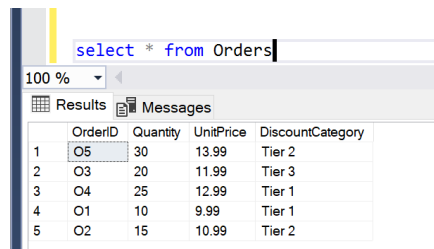
```
CREATE TABLE [dbo].[Orders]
(
    [OrderID] varchar(10), [Quantity] int, [UnitPrice] decimal(5,2), [DiscountCategory] varchar(10)
)
```

- Now to add an output in the dedicated sql pool just like in Data Factory we have to give a staging area, we go to storage account settings tab and give our storage account.
- Now we go to outputs and add this table as an output in the stream analytics. On the Add button in the outputs tab, select Azure Synapse Analytics. Give a name to it, select subscription, database, enter your credentials and the table name. Click save.
- You can test this output connection by clicking the test button right next to the delete button.
- Now go to the query part and enter the query –

```
SELECT OrderID,Quantity,UnitPrice,DiscountCategory INTO OrderSynapse
FROM programinput
```

- Replace ordersynapse and programinput with your own output and input alias. Save.
- Go to overview and start the job.
- Run the project on VS Studio

Data is loaded on to the Orders table –



	OrderID	Quantity	UnitPrice	DiscountCategory
1	O5	30	13.99	Tier 2
2	O3	20	11.99	Tier 3
3	O4	25	12.99	Tier 1
4	O1	10	9.99	Tier 1
5	O2	15	10.99	Tier 2

Stream analytics when starting gave us two options of Now and Custom. When we select Now it would only pick up events that are happening after Stream Analytics has started. This is the reason why previous events were not picked up.

When data is read by the stream analytics it can be seen in input preview tab of the query section of Stream analytics. Stream Analytics adds three columns of its own to the data –

EventProcessedUtcTime	PartitionId	EventEnqueuedUtcTime
"2021-10-29T21:43:21.7233342Z"	0	"2021-10-29T21:43:06.2430000Z"
"2021-10-29T21:43:21.7233342Z"	0	"2021-10-29T21:43:06.2430000Z"
"2021-10-29T21:43:21.7233342Z"	0	"2021-10-29T21:43:06.2430000Z"
"2021-10-29T21:43:21.7233342Z"	0	"2021-10-29T21:43:06.2430000Z"
"2021-10-29T21:43:21.7233342Z"	0	"2021-10-29T21:43:06.2430000Z"

This is the information that is taken by the Stream Analytics to understand the time of the event.

How to continuously stream data onto our stream events hub

Go to your adventureworks sql database that you had created. From the left hand menu, go to the diagnostic setting. This is where you can see all the different kinds of diagnostic settings that you can generate for this database.

- Start by going to your namespace and creating a new event hub with the message retention of 7 days.
- Go to the diagnostic setting page and click on Add Diagnostic Setting > select Basic under metric > On the right hand side select the destination as Event hub that we just created (if you are not able to see your namespace, it might be because they are not in the same region), name the setting at the top and click save.

Reading json file from a data lake storage account

Before we can connect our stream analytics job onto our newly created event hub. We should see how to read data from our data lake. The data that we would be reading would be the same kind of data that we would get from the diagnostic setting.

- Go to your stream analytics and inputs > Add stream input > Blob storage > Name it, select your storage account, connect via connection string, leave everything else as is, save.
- Go to query, choose the data lake input stream. Upon selection in the input preview tab it is showing the input data.
- In azure synapse, we create a table with the following query –

```
CREATE TABLE [dbo].[dblog] (  
[count] [bigint], [total] [bigint], [minimum] [bigint], [maximum] [bigint], [resourceId]  
[varchar](1000), [tame] [datetime2], [metricName] [varchar](500), [timeGrain] [varchar](100),  
[average] [bigint] )  
WITH  
(  
    DISTRIBUTION = ROUND_ROBIN,  
    HEAP  
)
```

- Create an output stream with the table that we just created above
- In the query section of Stream Analytics use the following query. Time is by default being read as nvarchar so I have to cast it in the next query

```
select [count],[total],[minimum],[maximum],[resourceId], [tame],[metricName],[timeGrain],[average]  
into jsonfile  
from  
(  
SELECT  
[count],[total],[minimum],[maximum],[resourceId], CAST([time] AS datetime) as  
[tame],[metricName],[timeGrain],[average]  
FROM blobinput  
) A  
where GetType( tame ) = 'datetime'
```

- Replace synapse and input with the output and the input streams named by you.
- Now to see the streaming in action, start the stream analytics
- After the stream analytics job has started only then, go to your data lake and create a new container and upload [this](#) file.
- Check the dblog table in synapse and the data should be there.

Side note : I just have to mention this. It took me a lot of time to debug this simple type error because the error message wasn't clear but I learned about different functions to find out the datatypes of the columns and whatnot.

Streaming data from an event hub that is getting diagnostics from a database

- Delete the data that is there in the synapse table that was used above.
- Go to the inputs of your stream analytics and add a new input stream of event hub. Name it, select your namespace, the event hub, connection string as authentication mode and everything as is and create.
- When we go to the query section for our stream analytics and see what is the data that is coming in, we see that the data is coming under a single column

[{"count":4,"total":0,"minimum":0,"maximum":0,"resourceId":"5051-10-30101323588818035","tame":"2010-10-30T01:33:28.1320000Z","metricName":"5051-10-30101323588818035","timeGrain":0,"average":0}]]	0	5051-10-30101323588818035	
[{"count":4,"total":0,"minimum":0,"maximum":0,"resourceId":"5051-10-30101323588818035","tame":"2010-10-30T01:33:28.1320000Z","metricName":"5051-10-30101323588818035","timeGrain":0,"average":0}]]	0	5051-10-30101323588818035	
[{"count":4,"total":0,"minimum":0,"maximum":0,"resourceId":"5051-10-30101323588818035","tame":"2010-10-30T01:33:28.1320000Z","metricName":"5051-10-30101323588818035","timeGrain":0,"average":0}]]	0	5051-10-30101323588818035	
records	EventProcessQueueTime	partitionId	EventEndQueueTime

- This is because the json data is in an array and then all the columns or json objects are inside a records property. We can see this by clicking the Raw button in the query section.

```
{
  "records": [
    {
      "count": 4,
      "total": 0,
      "minimum": 0,
      "maximum": 0,
      "resourceId": "/SUBSCRIPTIONS",
      "time": "2021-10-30T07:27:00.",
      "metricName": "cpu_percent",
      "timeGrain": "PT1M",
      "average": 0
    },
  ],
}
```

- So, we would have to modify our sql query as follows –

```
SELECT
  Records.ArrayValue.count as [count],
  Records.ArrayValue.total as [total],
  Records.ArrayValue.minimum as [minimum],
  Records.ArrayValue.maximum as [maximum],
  Records.ArrayValue.resourceId as [resourceId],
  CAST(Records.ArrayValue.time AS datetime) as [time],
  Records.ArrayValue.metricName as [metricName],
  Records.ArrayValue.timeGrain as [timeGrain],
  Records.ArrayValue.average as [average]
INTO
  OrderSynapse
FROM
  dbhub d
CROSS APPLY GetArrayElements(d.records) AS Records
```

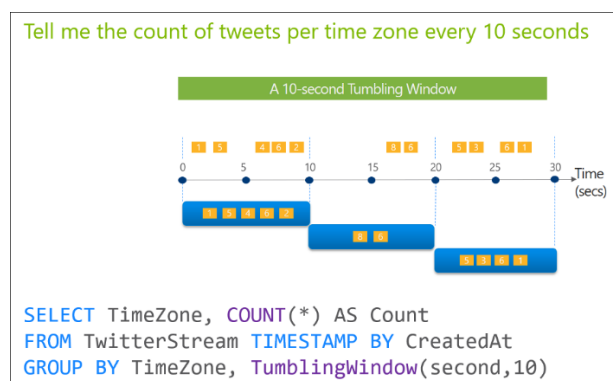
- We are using the GetArrayElements function that is available in Azure Stream Analytics. The following documentation was referred <https://docs.microsoft.com/en-us/stream-analytics-query/getarrayelements-azure-stream-analytics>
- We can select the query and click test query to see if it runs and gives the desired output
- Save the query
- Now we can go to overview and start.
- The data would start to appear in the table every few intervals would get added to the existing table

Windowing functions in Stream Analytics

Because the data is a constant stream, there may be periods of time for which we might want an aggregation. Let's say we could ask the stream to give the sum of a certain column every ten second.

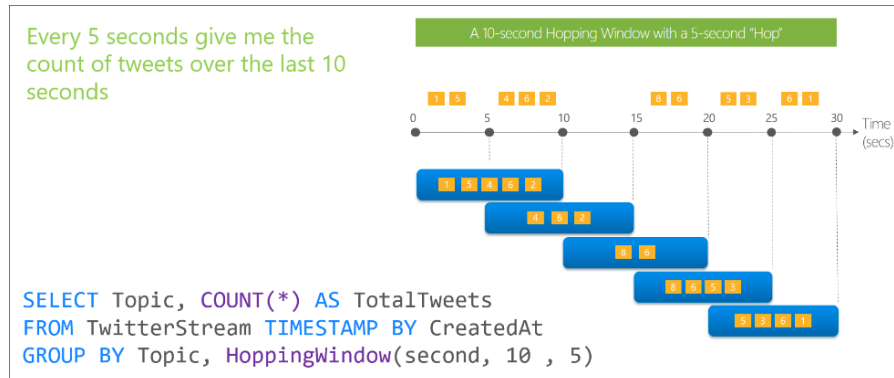
There are five different window functions –

1. Tumbling Window - 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.



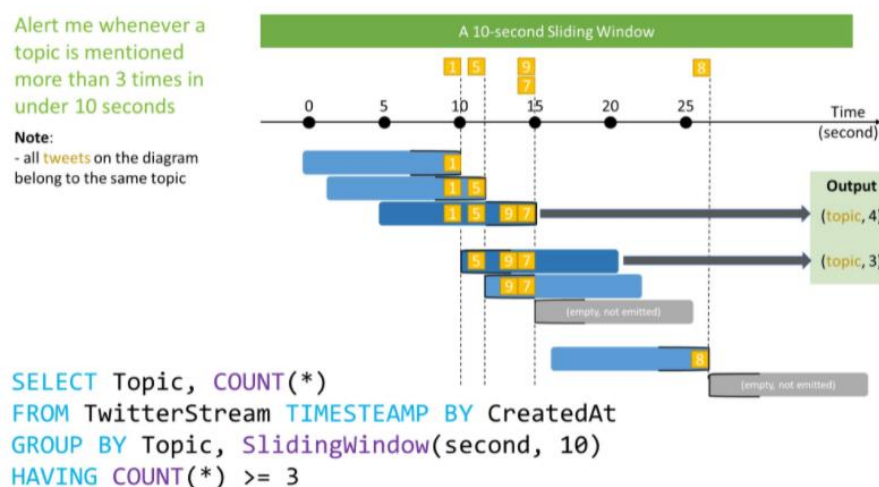
2. Hopping Window - Unlike tumbling windows, hopping windows model scheduled overlapping windows. A hopping window specification consist of three parameters: the timeunit, the window size (how long each window lasts) and the hop size (by how much each window moves forward relative to the previous one). 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.



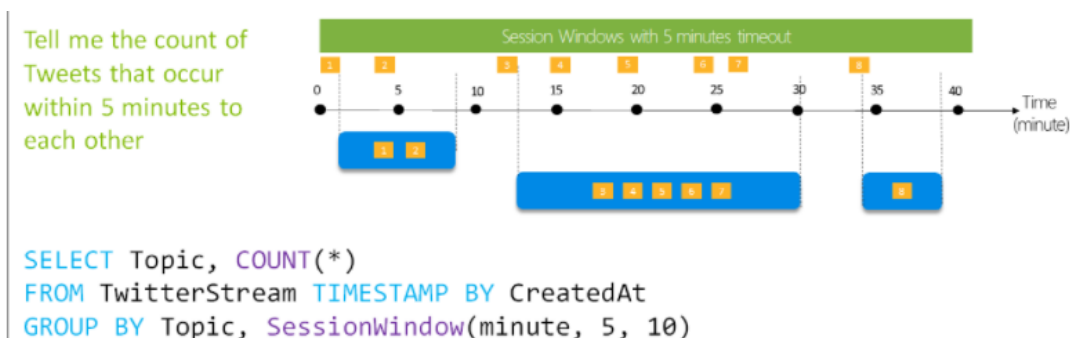
3. Sliding Window - When using a sliding window, the system is asked to logically consider all possible windows of a given length. As the number of such windows would be infinite, Azure Stream Analytics instead outputs events only for those points in time when the content of the window actually changes, 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.



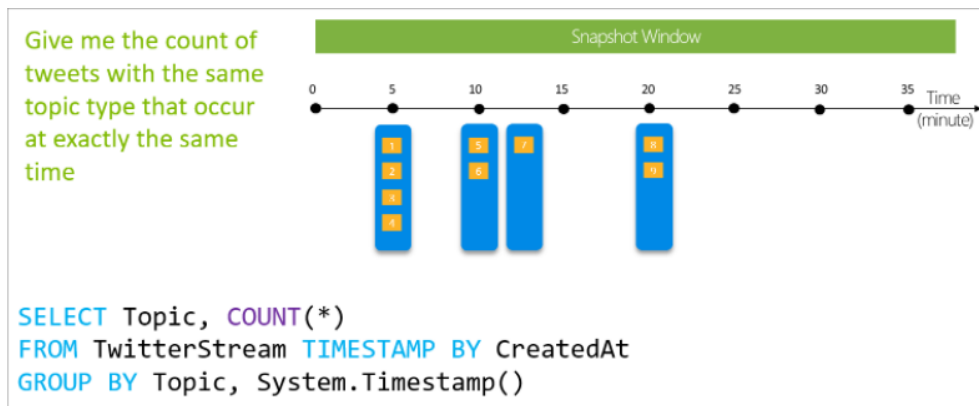
4. Session windows – Session Window group events that arrive at similar times, filtering out periods of time where there is no data. Session window function has three main parameters: timeout, maximum duration, and partitioning key (optional).

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.



5. Snapshot Window - Snapshot windows groups events that have the same timestamp. Unlike other windowing types, which require a specific window function (such as SessionWindow()), you can apply a snapshot window by adding System.Timestamp() to the GROUP BY clause.

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



We would be using the **tumbling window** function as an example. The code can be found in [this](#) file.

Now using the same stream as before, we are going to store the output in a summary table.

- Begin by creating the summary table with the query given in the file
- Now go to your stream analytics job, stop it if running. In the query section paste the query –

```
WITH
staginglogs AS
(
SELECT
Records.ArrayValue.metricName AS [metricName],
CAST(Records.ArrayValue.time AS datetime) AS [time]
FROM
    dbhub d TIMESTAMP BY EventEnqueuedUtcTime
    CROSS APPLY GetArrayElements(d.records) AS Records
)

SELECT staginglogs.metricName AS [metricName],COUNT(*) AS [Count],MAX(time) AS [TimeStamp]
INTO
    summary
FROM
    staginglogs
GROUP BY staginglogs.metricName,TumblingWindow(second,10)
```

- With the above query we are saying with the data that is coming in. Every ten seconds group the metric data by metric name, then get the count of the metrics and the max timestamp.
- Save the query, go to the outputs section and add a stream to the summary table that we created above.
- Now in the same stream analytics section, we could have multiple connection strings to the same event hub. This means with the same input stream we can populate the Summary table as well as the table that we were working on earlier to record the database metrics.
- Add the following query after the above query to populate the previous assignment table.

```
SELECT
Records.ArrayValue.count as [count],
Records.ArrayValue.total as [total],
Records.ArrayValue.minimum as [minimum],
Records.ArrayValue.minimum as [maximum],
```

```

Records.ArrayValue.resourceId as [resourceId],
CAST(Records.ArrayValue.time AS datetime) as [time],
Records.ArrayValue.metricName as [metricName],
Records.ArrayValue.timeGrain as [timeGrain],
Records.ArrayValue.average as [average]
INTO
    OrderSynapse
FROM
    dbhuball d
CROSS APPLY GetArrayElements(d.records) AS Records

```

- Now go to inputs and add another connection string to the same event hub to facilitate the stream to the above query.
- Delete everything from the dblog table previously used.
- Start the stream.
- We can see the summary table and the dblog table both are getting populated with the same stream

Pretty neat isn't it?

Reference Data

Sometimes in the Stream Analytics Job, the query might need to reference some static data that doesn't change much. In such cases, we can use reference data. If you want to perform a join of some incoming stream with your static data, in that case too we can make use of reference data.

The reference data could be in a storage account or it can be in an Azure SQL database.

- Begin by going to your storage account, create a new container and upload [this](#) file. This file contains the Metric name but also has tiers associated with different metrics.
- In the stream analytics inputs section, create a reference input for the file that we uploaded. Add reference input > Blob storage > give the name, select your storage account, connection string in authentication mode, enter the name of the file in path pattern and finally comma in the event serialization format.
- Delete the contents of the summary table
- No go to the query section and refer [this](#) script. The script is a variation of the above script and would populate the summary table but there's a join with the tier.csv file and the join condition is that only metric names with tier 2 would get populated.
- Start the stream analytics

We see the data is starting to populate which are tier 2 in the tier.csv file.