

## Agenda

### Introduction to Azure Stream Analytics

- Understanding Live Event Processing
- Azure stream analytics overview
- Create Azure stream Analytics Job

### Working with stream analytics job

- Understanding data stream input/output
- Create First stream analytics job using Blob storage
- Provision Azure Event Hub
- Configure Azure stream analytics using Event Hub and Blob Storage
- Extending Stream Topologies
- Using reference data

### Understanding Windowing Function and Timestamp

- Understanding Timestamp
- Understanding Windowing Function
  - Tumbling, Hopping, Sliding, Session Window

### Monitoring and optimising Stream analytics Job

## Understanding Live Event Processing

### What is streaming data?

- Streaming data refers to event data that is **continuously generated by sensors or other sources** and usually in **high volumes** and at **high velocity**.

Example:

- IoT sensors
- Click-stream data from apps and websites
- Social Media
- Server and security logs

### Need of stream analytics:

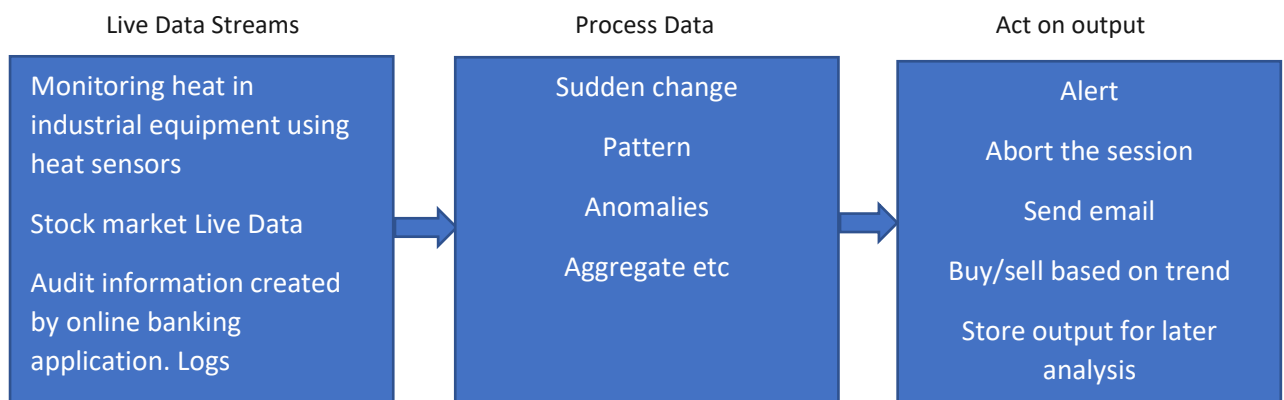
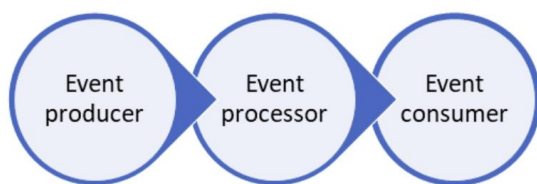
- Patterns and relationships can be identified in information extracted from different input sources.
- Information can be used Feed information to reporting tool or storing transformed data for later use.

- These patterns can be used to trigger some action like creating alert when certain thresholds are identified.
- To understand component or system behaviour under various conditions to fuel further enhancements of said component or system.

### Understanding Event Processing and Use Cases:

You can process the data streams with two approaches

1. On Demand: Streaming data can be collected over time and persisted in storage as static data and processed when convenient.
2. Live: Streaming data processed as it is ingested live.



### Azure Stream Analytics Overview

- Azure Stream Analytics is fully **managed**, **real time** analytics service designed to process **fast moving streams** of data from multiple sources simultaneously.

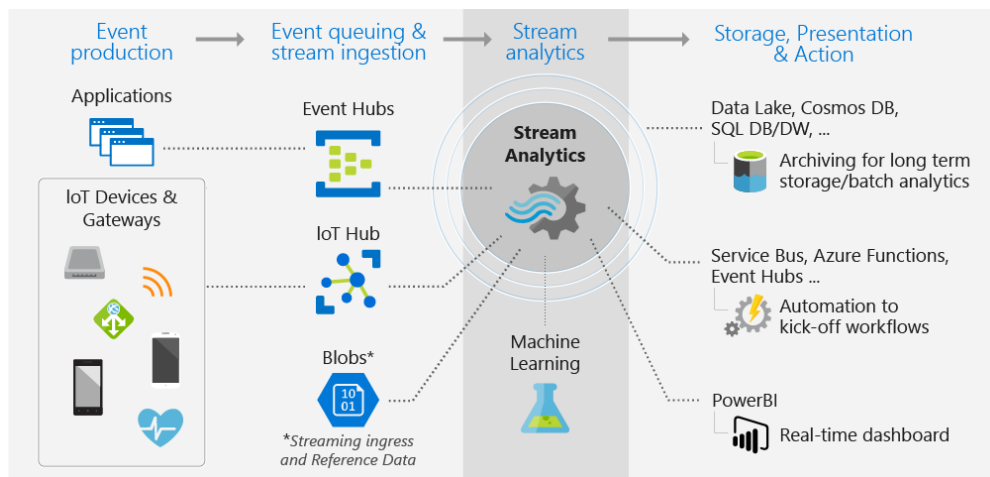
### Steps in Stream Analytics:

1. Capture and ingest the data
2. Process the data
3. Take some action, create report or store processed data in some storage

A typical event processing pipeline built on top of Stream Analytics consists of four components:

An event producer, an event ingestion system, the stream analytics engine, and finally the consumer.

Sources (Producer)→Ingestion→Analytics Engine→Destination (Consumer)



- Stream Analytics ingests data from Azure Event Hubs, Azure IoT Hub, or Azure Blob Storage.
- The input format supported by it is CSV, JSON, AvRo.
- It uses a simple SQL-based query language([Stream Analytics query language](#)) which can be used to easily filter, sort, aggregate, and join streaming data over a period of time. It is subset of Transact-SQL tailored to perform computations over streaming data
- You can also extend this SQL language with JavaScript and C# user defined functions (UDFs)
- You can easily adjust the event ordering options and duration of time windows when performing aggregation operations through simple language constructs and/or configurations.
- Azure Stream Analytics job can run in the cloud, for large-scale analytics, or run on IoT Edge for ultra-low latency analytics.
- Stream Analytics engine enables in-memory compute, it offers superior performance.

Create Azure stream Analytics Job

### Lab1: Provision Azure stream Analytics Job

Create Resource→Analytics→Stream Analytics Job→

Job name \*

Subscription \*

Resource group \*  
  
[Create new](#)

Location \*

Hosting environment ⓘ  
☒ Cloud ☐ Edge

Streaming units (1 to 192) ⓘ

→Create

### Pricing:

- Azure Stream Analytics is priced by the number of streaming units required to process the data into the service.
- Azure Stream Analytics on IoT Edge is pricing is based on job/device/month

Refer: <https://azure.microsoft.com/en-in/pricing/details/stream-analytics/>

## Understanding data stream input/output

- To create stream analytics job, you need to configure **Input** and **Output**.
- You need to write **Query** to process the data coming from Input and send it to Output.

### Stream Analytics Input

- Stream Analytics job can connect to one or multiple inputs.
- Input is connection string to existing data source.
- There are two Input Categories:
  1. Data Stream Input
  2. Reference Data Input

### Data Stream Input

- It is any data stream (ongoing sequence of events) which need to be processed in real time and acted upon.
- It is not a static data file and need to be generated constantly to have output from stream analytics
- You can have one or multiple inputs which you use in query.
- It can be from **Azure Event Hubs, IoT Hub or Blob storage**.
- Incoming data streams can be JSON, Avro, Csv

### Azure Event Hub:

- Event Hubs is a fully managed Platform-as-a-Service (PaaS) that you can use to ingest millions of events per second from devices or applications across the Internet.
- Client devices or any custom applications (web, mobile) can send messages about events to the event hub.
- Event Hubs acts as a front door for an event pipeline, where it receives incoming data and stores it until processing resources are available.

### Azure IoT Hub

- IoT Hubs are similar to event Hubs, but they're specifically designed and optimized for Internet of Things (IoT) scenarios.
- Using this you can accept events from thousands or millions of devices.
- The real key difference is one of the capabilities that an IoT hub has, which is to have **bidirectional communication**.
- Cloud service can generate messages and send them back to the devices. And those devices can take those messages and respond to them.
- For example: maybe restart a service or reset some configuration on the client device.

#### Note:

*When you write something to an IoT Hub there are a couple of columns are added to that, and one of those is this **EventEnqueuedUtcTime**. That's the time that the event was enqueued in the hub.*

### Azure Blob Storage

- It can be used as Input to stream analytics job.
- It is used for bulk data like log file.

Examples:

#### Event Hub

Custom stock trading application  
Online Banking Activities  
Data generated by Sensors, Devices  
Data generated by gaming engine

#### IoT Hub

Data generated by Sensors, Devices

#### Blob Storage

Online Banking Activities  
Log files from custom application

### Reference Data Input

- It can be the data which does not change or change very slowly, such as metadata lookup
- Azure stream analytics can accept reference data from Azure Blob storage and Azure SQL Database

Example:

- Metadata like Device details like name, capacity can be stored as reference data.
- List of registered devices
- Acceptable thresholds like allowed temperatures etc.

### Stream Analytics Output

- Outputs let you store and save the results of the Stream Analytics job, which further can be used for business analytics.
- In stream analytics query you need to refer to the name of the output by using the **INTO** clause.
- You can use a single output per job, or multiple outputs per streaming job) by providing multiple INTO clauses in the query.
- Output can be azure storage, data lake storage Datawarehouse's Database, PowerBI, CosmosDB etc

### Query

- Query is used for performing transformations and computations over streams of events.
- Stream Analytics query language is a subset of standard T-SQL syntax for doing Streaming computations.

Create First stream analytics job using Blob storage

### Lab2: First Stream Analytics job using Azure Blob storage as input

Create storage account (dssstreamsa)→Create Container "transactiondata"→Load "Withdrawals.json"

Configure Input

Job topology→Inputs→+Add Stream Input→Blob storage→

Blob storage

New input

Input alias \*

Withdrawals-Input

☐

Provide Blob storage settings manually

☒

Select Blob storage from your subscriptions

Subscription

Visual Studio Enterprise – VS

Storage account \* ⓘ

dssstreamsa

Storage account key

.....

Container \*

☐ Create new

☒

Use existing

transactiondata

Path pattern ⓘ

Date format

YYYY/MM/DD

Time format

HH

Partitions

1

Event serialization format \* ⓘ

JSON

Encoding ⓘ

UTF-8

Event compression type ⓘ

None

Configure Output

Blob storage/Data Lake Storage Gen2

New output

Output alias \*
Withdrawal-Output

☐ Provide storage settings manually
☒ Select storage from your subscriptions

Subscription
Visual Studio Enterprise – VS

Storage account \* ⓘ
dssstreamsa

Storage account key
.....

Container \*
☒ Create new ☐ Use existing
transactionoutput

Path pattern ⓘ
[date]

Date format
YYYY/MM/DD

Time format
HH

Event serialization format \* ⓘ
JSON

Encoding ⓘ
UTF-8

Format ⓘ
Line separated

Minimum rows ⓘ
50

Maximum time
Hours ⓘ
Minutes
5

Authentication mode
Connection string

Save

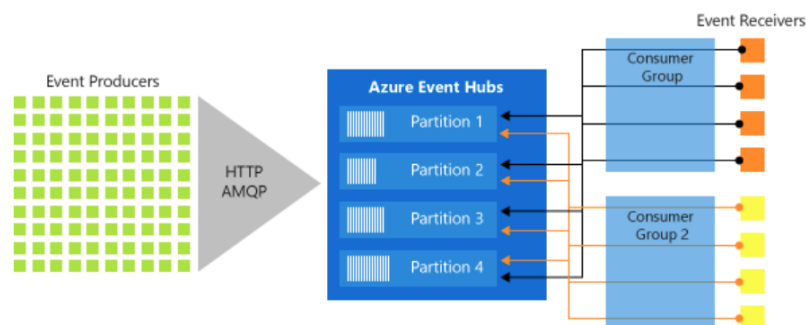
## Configure Query

SELECT  *  INTO [Withdrawal-output]  FROM [Withdrawal-Input]	SELECT  deviceId, cardNumber, amount  INTO [Withdrawal-output]  FROM [Withdrawal-Input]
--	---



- Event Hubs is a fully managed Platform-as-a-Service (PaaS) that you can use to ingest millions of events per second from devices or applications across the Internet.
- Client devices or any custom applications (web, mobile) can send messages about events to the event hub.
- For Event Hub first you need to create namespace and, in that namespace, you can create up to 10 event hubs.
- Namespace is logical container for multiple event hubs, each event hub represents unique stream of data. It is scoping container having multiple shared properties (throughput, cost).
- Entire namespace can be secured using **Shared access signature key**
- These keys can be used to give permission like send, manage and receive to event hub.
- In event Hub you can create up to 32 partitions so that workload can be processed in scalable manner.
- The number of partitions in an event hub directly relates to the number of concurrent readers you expect to have.
- Messages sent to event hub can be distributed in Round Robin across the partitions or you can specify the partition key.
- Messages can be archived to blob storage
- Event Hubs retains data for a configured retention time that applies across all partitions in the event hub

#### Event Hubs stream processing architecture:



#### Publishers/Producers

- An entity that sends data to the Event Hubs is called a ***publisher***. Event publishers are any application or device that can send out events using either HTTPS or Advanced Message Queuing Protocol (AMQP) 1.0, Kafka 1.0 and later.
- Event publishers use a Shared Access Signature (SAS) token to identify themselves to an event hub, and can have a unique identity, or use a common SAS token.

#### Subscribers /Consumer

- An entity that reads data from the Event Hubs is called a **consumer** or a **subscriber**
- Azure Event Hubs makes sure that you can receive events from variety of sources, fast, in -order store it reliably and durably, has multiple consumers and consumer groups for quick and concurrent data processing.

#### Consumer groups

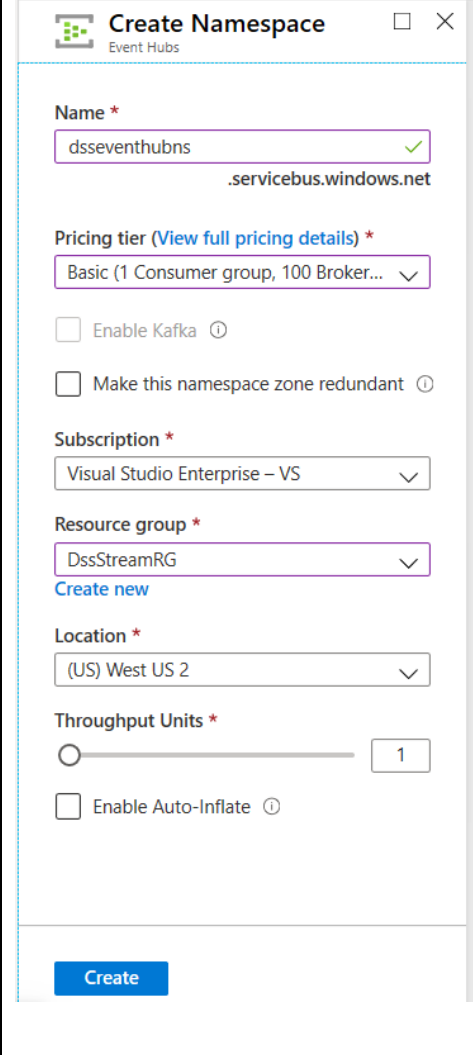
- An Event Hub **consumer group** represents a specific view of an Event Hub data stream.
- By using separate consumer groups, multiple subscriber applications can process an event stream independently, and without affecting other applications, at their own pace and with their own offsets.
- In a stream processing architecture, each downstream application equates to a consumer group. If you want to write event data to long-term storage, then that storage writer application is a consumer group.

### Provision Azure Event Hub

#### Lab3: Provision Event Hub

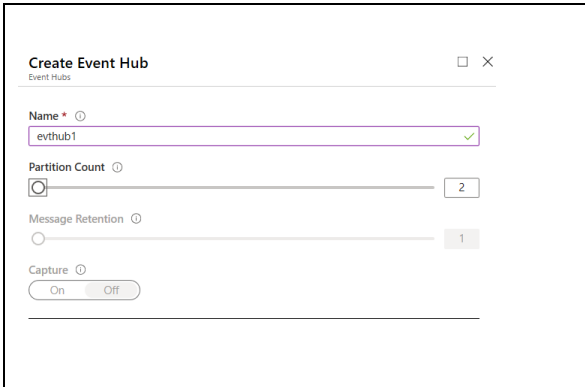
1. Create Event Hub Namespace

Create a resource → Search Event Hub → Select → Create →

	<p>Pricing Tier: Basic/Standard</p> <p>Consumer Groups: How many unique applications Reading entire stream of data.</p> <p><b>Throughput Units:</b> It is performance unit. How many messages can be processed by event hubs</p> <p><b>Enable Auto-inflate:</b> Autoscale throughput units</p>
--	--

## 2. Create Event Hub

Entities→Event Hubs→+Event Hub

	<p><b>Partition Count:</b> 1 to 32</p> <p><b>Message Retention:</b> No of days to keep message in event hub for processing.</p> <p><b>Capture:</b> Enables you to automatically deliver the streaming data in Event Hubs to an Azure Blob storage or Azure Data Lake Store, with the added flexibility of specifying a time or size interval.</p>
---	---

→Create

## 3. Create Event Hub for Temperature Data

Entities→Event Hubs→+Event Hub→  
Name:TempDataHub01-->  
Partition Count:2  
→Create

#### Shared Access Policy in Event Hub:

- SAS provides you way to grant limited access of resources in your event hubs namespace based on authorization rules.
- You can generate SAS key token using portal or code.
- A client application can then pass the token to Event Hubs to prove authorization for the requested operation.

#### Lab4: Use demo program used to generate temperature and sending to Event Hub

1. Create credentials for event hub

Settings→Shared Access Policy→Add→

→Create

You can Use these keys with any client application which need to send events to these event hub namespace

2. Use demo program used to generate temperature and sending to Event Hub
3. Open already existing program and Verify if Newget package for Event Hub is installed  
Solution Explorer→Project name→Right Click→Manage NueGet Packages→Verify in Install tab  
You can install using browse tab, if package is not installed.

4. Use Event Hub Connection string and Event Hub Name in program

5. Connection string:

Endpoint=sb://dsseventhubs.servicebus.windows.net/;SharedAccessKeyName=TempData;SharedAccessKey=0Ndx/ZBS0Hw0RtTRBrYeC35gRDb6bdjMPN7NOdFW5A4=

6. Modify following code with your key

```
private const string EventHubConnectionString =  
    "Endpoint=sb://dsseventhubs.servicebus.windows.net/;" +
```

```
"SharedAccessKeyName=TemperatureData;SharedAccessKey=gkjTph9KHsNvI/wVO+7I7wY
QcfhPewOhkloEjjzKq8E=";
private const string EventHubName = "tempdatahub01";
```

## Configure Azure stream analytics using Event Hub and Blob Storage

### Lab5: Configure Azure stream analytics using Event Hub and Blob Storage

Create Resource → Analytics → Stream Analytics Job →

Job name \*  
ProcessEventHubTemperaturedata ✓

Subscription \*  
Visual Studio Enterprise – VS

Resource group \*  
DssStreamRG  
[Create new](#)

Location \*  
(US) Central US

Hosting environment ⓘ  
☒ Cloud ☐ Edge

Streaming units (1 to 192) ⓘ  
 1

→ Create

### Configure Input

Job Topology → Inputs → +Stream Input → Event Hub →

Event Hub

New input

Input alias \*

TemperatureEvthub-Input

☐

Provide Event Hub settings manually

☒

Select Event Hub from your subscriptions

Subscription

Visual Studio Enterprise – VS

Event Hub namespace \* ⓘ

dsseventhubs

Event Hub name \* ⓘ

☐ Create new

☒

Use existing

tempdatahub01

Event Hub policy name \* ⓘ

☐ Create new

☒

Use existing

TemperatureData

Event Hub policy key

\*\*\*\*\*

Event Hub consumer group \* ⓘ

☐ Create new

☒

Use existing

\$Default

Event serialization format \* ⓘ

JSON

Encoding ⓘ

UTF-8

Event compression type ⓘ

None

→Save

## Configure Output

## Blob storage/Data Lake Storage Gen2

New output

Output alias \*

TemperatureBlob-output ✓

☐ Provide storage settings manually

☒ Select storage from your subscriptions

Subscription

Visual Studio Enterprise – VS

Storage account \* ⓘ

dssstreamsa

Storage account key

.....

Container \*

☒ Create new ☐ Use existing

temperaturesdata ✓

Path pattern ⓘ

output/{date} ✓

Date format

YYYY/MM/DD

Time format

HH

Event serialization format \* ⓘ

JSON

Encoding ⓘ

UTF-8

Format ⓘ

Line separated

Minimum rows ⓘ

10 ✓

Maximum time

Hours ⓘ

Minutes

0

## Blob storage/Data Lake Storage Gen2

New output

Output alias \*

tempbloboutput02 ✓

☐ Provide storage settings manually

☒ Select storage from your subscriptions

Subscription

Visual Studio Enterprise – VS

Storage account \* ⓘ

dssstreamsa

Storage account key

.....

Container \*

☒ Create new ☐ Use existing

output01 ✓

Job Topology→Outputs→

Job Topology→Query→

```

SELECT
    inp.TemperatureCelcius,inp.SensorId,inp.EventEnqueuedUtcTime
INTO
    [TemperatureBlob-output]
FROM
    [TemperatureEvthub-Input] AS inp

```

→Save Query

*Note: In query you can retrieve following metadata fields, when data comes from Event Hub stream input.*

**EventEnqueuedUtcTime** is the timestamp of an event's arrival in an event hub and is the default timestamp of events coming from Event

Hubs to Stream Analytics

**EventProcessedUtcTime** is the date and time that the event was processed by stream Analytics

### Start Job Options

There are three options available for job start

1. **Now:** Makes the starting point of the output event stream the same as when the job is started.
2. **Custom:** You can choose the starting point of the output.
3. **When last stopped.** This option is available when the job was previously started, but was stopped manually or failed.

**For all options** Azure Stream Analytics will automatically read the data prior to this time if a temporal operator is used.

## Extending the stream Topologies

You can have multiple Inputs, outputs and have multiple queries running.

### Lab6:

**Create a job consuming temperature data from event hub and outputting all data to blob storage and High-temperature data to another event hub.**

**Create one more job to process data from second event hub and output it to blob storage**

1. Create one more event Hub Instance("**Hightempevthub**") in same namespace.
2. Add one more output(**Hightempevthub-output**) as event hub to existing stream analytics job
3. Change the query, using 2 outputs as follows:

```

WITH [AllTempReardings] AS
(SELECT * FROM [TemperatureEvthub-Input])

```



SELECT

TemperatureCelcius, SensorId

INTO

[TemperatureBlob-output]

FROM

[AllTempReadings]

SELECT

TemperatureCelcius, SensorId

INTO

[Hightempevthub-output]

FROM

[AllTempReadings]

Where TemperatureCelcius > 200

4. Create new job(Alerthightemp)
5. Configure Input from “**hightempevthub**”
6. Configure Output(HighTempalertBlob-Output) as Blob storage and create the file in Hierarchical fashion

Blob storage/Data Lake Storage Gen2

×

New output

Output alias \*

HighTempalertBlob-Output ✓

☐ Provide storage settings manually
 ☒ Select storage from your subscriptions

Subscription

Visual Studio Enterprise – VS ▾

Storage account \* ⓘ

dssstreamsa ▾

Storage account key

.....

Container \*

☒ Create new
 ☐ Use existing

temp-alert ✓

Path pattern ⓘ

alerts/{date} ✓

Date format

YYYY/MM/DD ▾

Time format

HH ▾

Event serialization format \* ⓘ

JSON ▾

Encoding ⓘ

UTF-8 ▾

Format ⓘ

Line separated ▾

Minimum rows ⓘ

Maximum time

Hours ⓘ

Minutes

Authentication mode

Connection string ▾

## Query

SELECT

\*

INTO

[HighTempalertBlob-Output]

FROM

[Hightempevthub-Input]

## Reference Data

Stream Analytics can have input from sources generating real time data

- It can also have input which can be static data called reference data.
- Reference data can be used to lookup the values which doesn't change over a period of time.
- You can use query to combine data from real -time stream and reference data,using join



**Lab 7: Use the static file in blob storage containing sensor details. Use the information in query.**

1. Load the Sensor.Csv File in blob storage
2. Create reference input for this file
3. Change the query as follows

**SELECT**

ht.\*,br.SensorName

**INTO**

[HighTempAlertBlob-Output]

**FROM**

[Hightempevthub-Input] as ht

**JOIN** [blobrefdata] as br

ON ht.SensorId =br.SensorId

## Understanding Timestamp

- Timestamps are an important concept when it comes to dealing with events that occur along a timeline.
- Every piece of data coming in to azure stream analytics has timestamp which is used to process the data over time.
- By default, the timestamp foreach event that occurs in our Azure stream analytics query is based on the source that it came from (Arrival Time).

Event Hub	Event arrival time in event Hub, IoT Hub	
Blob Storage	Last modified time of blob	

Sample Events:

ID	Time	Val		ID	Time	Val	
SEN-001	00:01	200		SEN-002	00:02	427	

Sample Query:

```
SELECT id,System.Timestamp,Val
```

```
INTO output FROM input
```

### System.Timestamp()

Event timestamp can be retrieved in the SELECT statement in any part of the query using **System.Timestamp()** property.

### Event Time Vs Arrival Time

- **Event time** is the time at which the **event is generated**.
- Event **arrival time** is the time is time when the event is **ingested**.
- If you want to use Event time instead of Arrival time while processing the event, you can use **TIMESTAMP BY** clause to specify custom timestamp values and timestamp can be the field in actual event.
- If a **TIMESTAMP BY** clause is not specified for a given input, arrival time of the event is used as a timestamp.

Example:

```
Select AlertTime,Temperature,ValveNumber
```

```
From input TIMESTAMP BY AlertTime
```

- Using Custom time can have issue of out of order of events (late messages)

Note: You can access arrival time by using the **EventEnqueuedUtcTime** property for Event Hubs inputs, **IoTHub.EnqueuedTime** property for IoT Hub, and using the **BlobProperties.LastModified** property for blob input.

## Understanding Windowing Functions

- Each data event has timestamp, by default its value is time when the event is ingested.
- Stream input is potentially infinite stream and aggregated values are possible when limiting timespan over which they are computed.
- In real time event processing you need to perform aggregation over subset of events that fall within some period of time (time window).

- The concept of time is a fundamental necessity to complex event-processing systems.

Example:

## Time Windowing

Name	Date	TimeOff
John	Mon	1.5
Mary	Tue	3.0
Reza	Tue	2.0
John	Fri	3.0

Give me total hourly time off per employee

```
SELECT Name, SUM(TimeOff)
FROM Employee
GROUP BY (Name)
```

```
John    4.5
Mary    3.0
Reza    2.0
```

Timestamp	Sensor	Temperature
10:12:34:00	SEN01	97.5
10:12:34:05	SEN01	99.2
10:12:34:10	SEN01	120.4
10:12:34:15	SEN01	170.6
10:12:34:20	SEN01	180.9
10:12:34:25	SEN01	195.0
10:12:34:30	SEN01	200.5



Give me the average SEN01 temperature for the past 10 seconds

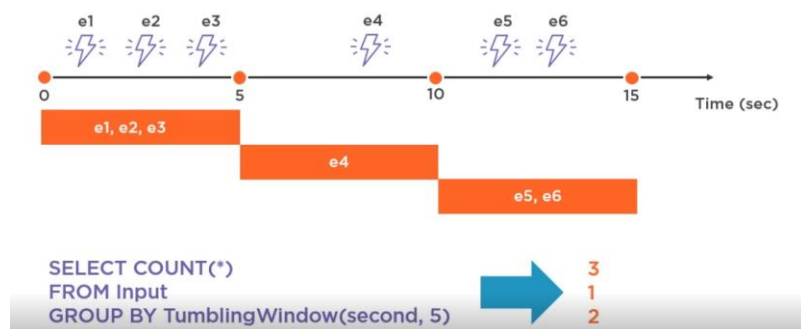
Stream Analytics has native support for windowing functions, enabling developers to author complex stream processing jobs with minimal effort.

### Types of Windows

1. Tumbling
2. Hopping
3. Sliding
4. Session

### Tumbling Window:

Size of window is **fixed**, **no overlapping** between consequent windows.



### Syntax:

**Tumbling window(timeunit>windowsize,[offset])**

Timeunit:

- Can be day(dd,d),hour(hh),minute(mi,n),second(ss,s),millisecond(ms),microsecond(mcs)

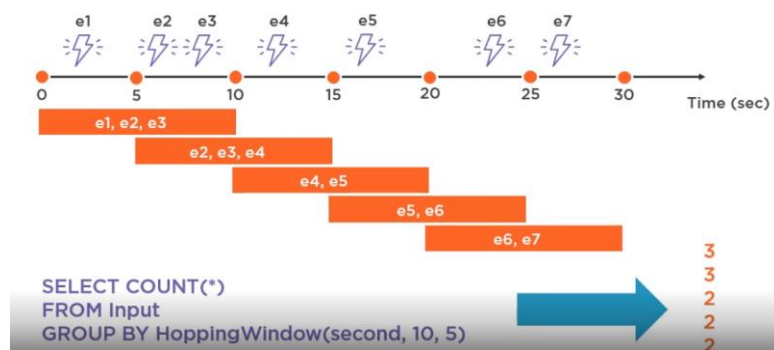
Windowsize:

- The maximum size of the window is 7 days.

```
SELECT CAST( AVG(TemperatureCelcius) AS bigint) AS AverageTemp,SensorId
INTO Output1
FROM HeatData
GROUP BY TumblingWindow(second,5),SensorId
```

### Hopping Window:

Size of window is **fixed**, but **overlapping** between consequent windows.



### Syntax:

Tumbling window(timeunit>windowsize,hopsize,[offset])

Timeunit :

- Can be day(dd,d),hour(hh),minute(mi,n),second(ss,s),millisecond(ms),microsecond(mcs)

Windowsize:

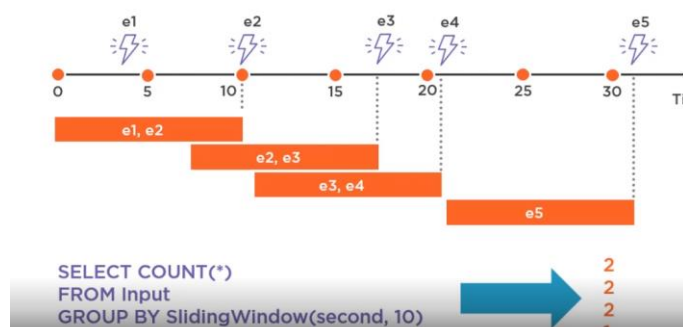
- The maximum size of the window is 7 days.

Hopsize:

- A big integer describes the size of hop(how much window should move relative to previous).

### Sliding Window:

Size of window is **fixed**, but new window will be created when new event happens. **Windows may overlap**



### Syntax: SlidingWindow(timeunit,windowsize)

Timeunit:

- Can be day(dd,d),hour(hh),minute(mi,n),second(ss,s),millisecond(ms),microsecond(mcs)

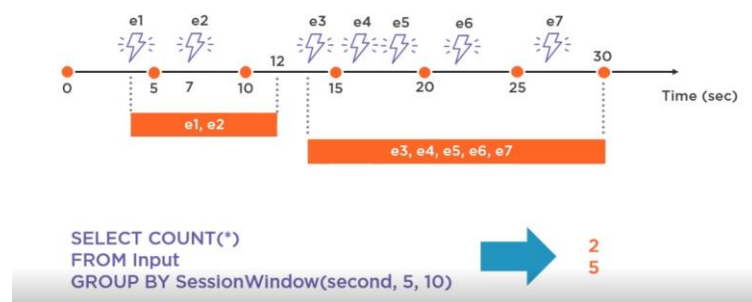
Windowsize:

- The maximum size of the window is 7 days.

### Session Window:

- Size of window is **not fixed** and it is **not overlapping**
- A session window begins when the first event occurs and captures all events which happens till **Timeoutsize**.
- If another event occurs within the specified timeout from the last ingested event, then the window extends to include the new event.  
If no events occur within the timeout, then the window is closed at the timeout.
- If events keep on happening before timeout, window will get extended limitless. To prevent this **maxDurationSize** is specified
- There can be period of silence when there is no event and there is no window
- Size of session window is checked only at multiple of **maxdurationsize**.
- Session windows group events that arrive at similar times, filtering out periods of time where there is no data.

Example: Group and count all events occurring at 5 min to each other



### Syntax: Session window(timeunit,timeoutsize,maxdurationsize,[offset])

Timeunit :

- Can be day(dd,d),hour(hh),minute(mi,n),second(ss,s),millisecond(ms),microsecond(mcs)

Timeoutsize:

- A big integer that describes the gap size of the session window. Data that occur within the gap size are grouped together in the same window
- If the total window size exceeds the specified maxDurationSize at a checking point, then the window is closed and a new window is opened at the same point.

## Sample Queries

### Lab 8: Find average temperature of each sensor in every 10 seconds (Tumbling)

Use "ProcessEventHubTemperatureData" Job → Inputs → Sample data from input → Select start time and duration → Sample → Download Sample

Use following query:

```
SELECT DateAdd(second,-10,System.Timestamp) as Winstart,
System.Timestamp AS WinEnd,
    Avg(TemperatureCelcius) AS Avgtemperature,SensorId
FROM
    [AllTempReardings]
GROUP BY SensorId,SlidingWindow(second,10)
```

### Lab 9: Find average temperature of each sensor in last 10 second every 5 second (Hopping)

```
SELECT
    Avg(TemperatureCelcius) AS Avgtemperature,SensorId
INTO
    [TemperatureBlob-output]
FROM
    [AllTempReardings]
GROUP BY SensorId
,Hoppingwindow(second,10,5)
```

### Lab 10: Find number of transactions that occurred in 1 min interval. (Tumbling)

Use dssstreamjob-1 → Use the sample provided → Modify query and test

```
SELECT System.Timestamp as [Time Ending],
    COUNT(*) AS [Number of Transactions]
FROM Withdrawals TIMESTAMP BY TransactionTime
GROUP BY TumblingWindow(n, 1)
```

### Lab 11 :Identify fraud transaction(transactions involving the same ATM card but different ATM machines that take place within 60 seconds of each other.)

```
SELECT W1.CardNumber as [Card Number],
```



```

W1.DeviceID as [ATM 1], W2.DeviceID as [ATM 2],
W1.TransactionTime as [Time 1], W2.TransactionTime as [Time 2]
FROM [Withdrawals-Input] W1 TIMESTAMP BY TransactionTime
JOIN [Withdrawals-Input] W2 TIMESTAMP BY TransactionTime
ON W1.CardNumber = W2.CardNumber
AND DATEDIFF(ss, W1, W2) BETWEEN 0 and 60
WHERE W1.DeviceID != W2.DeviceID

```

### Event Ordering Policy

#### **Late arrival policy:**

- Compare the Timestamp of event with current time. If it is within policy window adjust timestamp for that event or drop the event, otherwise accept the event automatically.

Example:

Event Time 00:10:00, Arrival Time 00:10:40, Late arrival policy :15 sec.

Event time adjustment: (arrival time - late arrival policy value) =00:10:25

#### **Out of order policy:**

- After event time is adjusted based on late arrival policy, you can also choose to automatically drop or adjust events that are out-of-order.

If you set this policy to 8 seconds, any events that arrive out of order but within the 8-second window are reordered by event time. Events that arrive later will be either dropped or adjusted to the maximum out-of-order policy value.

Refer: <https://docs.microsoft.com/en-us/azure/stream-analytics/event-ordering#what-is-late-arrival-policy>

## Optimize and Monitor Stream Analytics Job

#### **Streaming Units (SU):**

- Streaming Units represents the computing resources that are allocated to execute a Stream Analytics job.
- The higher the number of SUs, the more CPU and memory resources are allocated for your job
- Azure Stream Analytics jobs perform all processing in memory. When running out of memory, the streaming job fails.
- The SU % utilization metric, which ranges from 0% to 100%, describes the memory consumption of your workload.
- It's best to keep the SU metric below 80% to account for occasional spikes. Microsoft recommends setting an alert on 80% SU Utilization metric to prevent resource exhaustion

Note: By default, each Azure subscription has a quota of up to 500 SUs for all the analytics jobs in a specific region

### Query Data Using Azure Stream Analytics

- Azure stream Analytics queries are written using stream analytics query language.
- It is similar to T-SQL which maps input streams into stream analytics output.
- Azure stream analytics can take input from Event Hub, IoT Hub, Blob storage.
- The input format supported by it is CSV, JSON, AvRo.

### Supported Data Types

- Bigint, float, nvarchar(max), datetime, bit, record, array
- Record is Key value pair with supported data types
- Array is list of values of supported data type.

### Type Casting

- You have different casting functions like CAST, TRY\_CAST
- Type cast error happening during input read causes the event to drop
- Type cast error happening during output write will be handled by error policy.