**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

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.

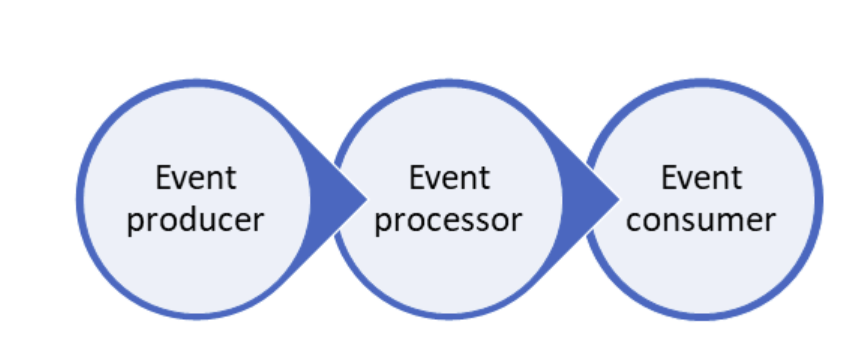
**Example:**

* Online stores analyzing real-time clickstream data to provide product recommendations to consumers as they browse the website.
* Manufacturing facilities using telemetry data from IoT sensors to remotely monitor high-value assets.
* Credit card transactions from point-of-sale systems being scrutinized in real-time to detect and prevent potentially fraudulent activities.

**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.



Live Data Streams Process Data Act on output

Monitoring heat in industrial equipment using heat sensors

Stock market Live Data

Audit information created by online banking application. Logs

Alert

Abort the session

Send email

Buy/sell based on trend

Store output for later analysis

Sudden change

Pattern

Anomalies

Aggregate etc

**Stream Processing Solution Characteristics:**

A screenshot of a computer

Description automatically generated with low confidence

1. The source data stream is *unbounded* - data is added to the stream perpetually.
2. Each data record in the stream includes *temporal* (time-based) data indicating when the event to which the record relates occurred (or was recorded).
3. Aggregation of streaming data is performed over temporal *windows* - for example, recording the number of social media posts per minute or the average rainfall per hour.
4. The results of streaming data processing can be used to support real-time (or *near* real-time) automation or visualization, or persisted in an analytical store to be combined with other data for historical analysis. Many solutions combine these approaches to support both real-time and historical analytics.

**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.

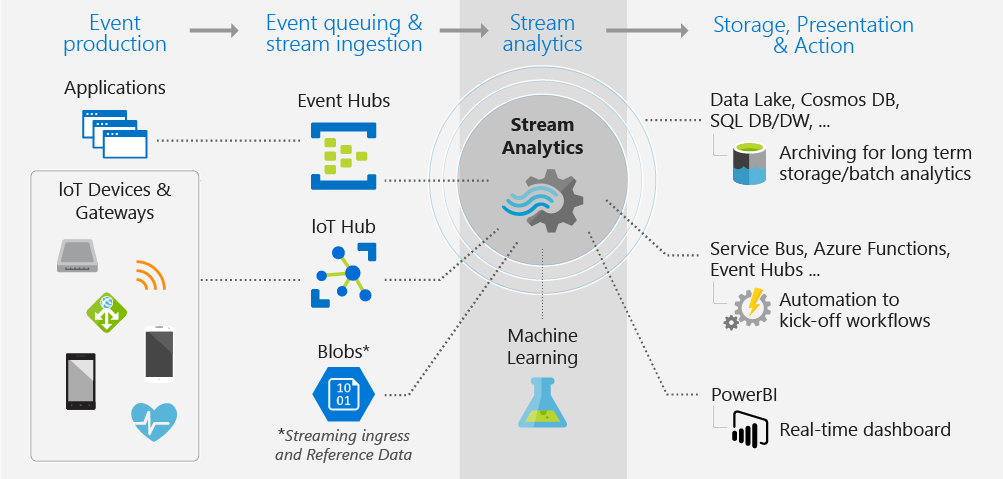
**Steps in Stream Analytics:**

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)

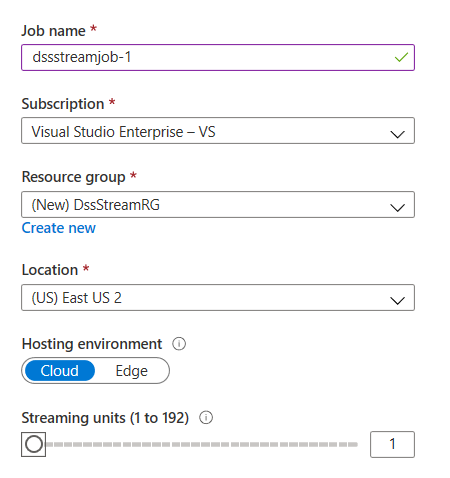


* Stream Analytics ingests data from Azure Event Hubs, Azure IoT Hub, or Azure Blob Storage.
* The input format supported by it is CSV, JSON, AvRo.
* It uses a simple SQL-based query language([Stream Analytics query language](https://docs.microsoft.com/stream-analytics-query/stream-analytics-query-language-reference)) 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 preforming 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🡪

**🡪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 torefer to the name of the output by using the [**INTO** clause](https://docs.microsoft.com/stream-analytics-query/into-azure-stream-analytics).
* You can use a single output per job, or multiple outputs per streaming job) by providing multiple INTO clauses in the query.
* It supports a wide range of outputs, which can be used to:
  + Persist the results of stream processing for further analysis; for example by loading them into a data lake or data warehouse.
  + Display a real-time visualization of the data stream; for example by appending data to a dataset in Microsoft Power BI.
  + Generate filtered or summarized events for downstream processing; for example by writing the results of stream processing to an event hub.
* 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.

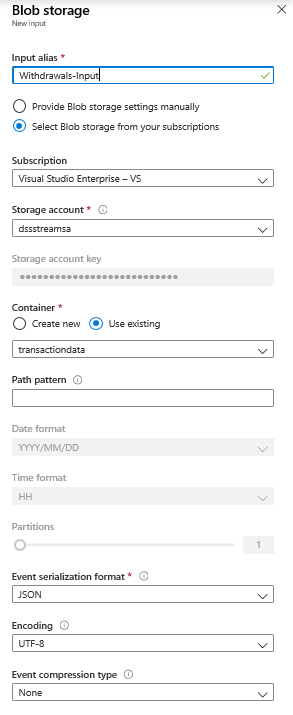
Create First stream analytics job using Blob storage

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

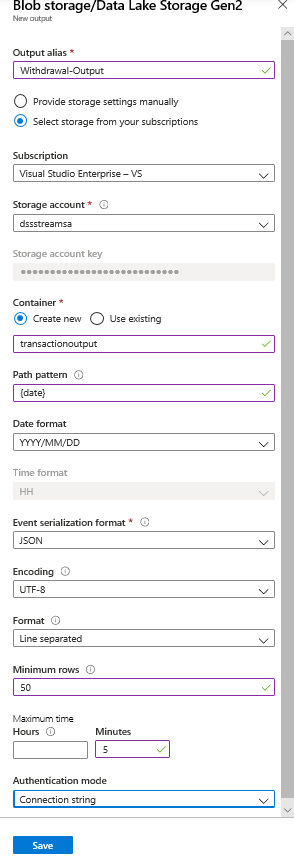
Create storage account (dssstreamsa)🡪Create Container “transactiondata”🡪Load “Withdrawals.json”

Configure Input

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



Configure Output



Configure Query

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

**Lab 2 :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

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

Diagram

Description automatically generated

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 | | SEN-001 | 00:01 | 200 | | |  |  |  | | --- | --- | --- | | ID | Time | Val | | 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,Temprature,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 is 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:

Table

Description automatically generated

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

Types of Windows:

1. [**Tumbling**](https://docs.microsoft.com/stream-analytics-query/tumbling-window-azure-stream-analytics)
2. [**Hopping**](https://docs.microsoft.com/stream-analytics-query/hopping-window-azure-stream-analytics)
3. [**Sliding**](https://docs.microsoft.com/stream-analytics-query/sliding-window-azure-stream-analytics)
4. [**Session**](https://docs.microsoft.com/stream-analytics-query/session-window-azure-stream-analytics)
5. **Snapshot**

**Tumbling Window:**

Size of window is **fixed**, **no overlapping** between consequent windows. Events can't belong to more than one tumbling window.

A screenshot of a computer

Description automatically generated with low confidence

**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 DateAdd(minute,-1,System.TimeStamp) AS WindowStart,

System.TimeStamp() AS WindowEnd,

MAX(Reading) AS MaxReading

INTO

[output]

FROM

[input] TIMESTAMP BY EventProcessedUtcTime

GROUP BY TumblingWindow(minute, 1)

**Hopping Window:**

Size of window is **fixed, but overlapping** between consequent windows, jumping forward in time by a fixed period.

A picture containing text, screenshot, font, line

Description automatically generated

**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).

Example: outputs an event every 30 seconds containing the maximum reading value that occurred over the last 60 seconds.

SELECT DateAdd(second,-60,System.TimeStamp) AS WindowStart,

System.TimeStamp() AS WindowEnd,

MAX(Reading) AS MaxReading

INTO

[output]

FROM

[input] TIMESTAMP BY EventProcessedUtcTime

GROUP BY HoppingWindow(second, 60, 30)

**Sliding Window:**

Size of window is **fixed**, but new window will be created when content of the window actually changes. Azure Stream Analytics outputs events for only those points in time when an event entered or exited the window. As such, every window contains a minimum of one event **Windows may overlap**

A picture containing text, screenshot, font, line

Description automatically generated

**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.

Example: Find maximum reading value in each one-minute window in which an event occurred.

SELECT DateAdd(minute,-1,System.TimeStamp) AS WindowStart,

System.TimeStamp() AS WindowEnd,

MAX(Reading) AS MaxReading

INTO

[output]

FROM

[input] TIMESTAMP BY EventProcessedUtcTime

GROUP BY SlidingWindow(minute, 1)

**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.

A screenshot of a computer

Description automatically generated with medium confidence

**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.

Example: Measures user session length by creating a SessionWindow over clickstream data with a timeoutsize of 20 seconds and a maximumdurationsize of 60 seconds.

SELECT DateAdd(second,-60,System.TimeStamp) AS WindowStart,

System.TimeStamp() AS WindowEnd,

MAX(Reading) AS MaxReading

INTO

[output]

FROM

[input] TIMESTAMP BY EventProcessedUtcTime

GROUP BY SessionWindow(second, 20, 60)

**Snapshot Window:**

**Snapshot** windows groups events by identical timestamp values. Unlike other windowing types, a specific window function isn't required. You can employ a snapshot window by specifying the System.Timestamp() function to your query's GROUP BY clause.

A screenshot of a computer

Description automatically generated with low confidence

Example: finds the maximum reading value for events that occur at precisely the same time.

SELECT System.TimeStamp() AS WindowTime,

MAX(Reading) AS MaxReading

INTO

[output]

FROM

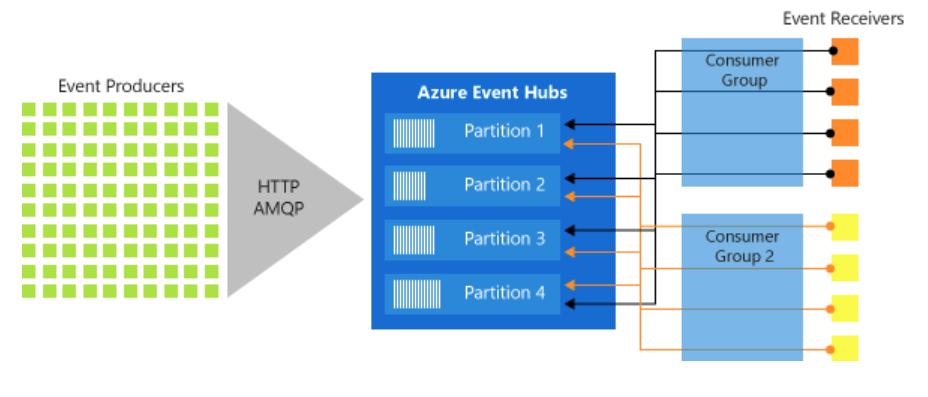
[input] TIMESTAMP BY EventProcessedUtcTime

GROUP BY System.Timestamp()

Introduction To 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.
* 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 (throuput, 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🡪

Graphical user interface, text, application, email

Description automatically generated🡪Review+Create

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

1. Create Event Hub

Entities🡪Event Hubs🡪+Event Hub

Graphical user interface, text, application

Description automatically generated

🡪Next

Graphical user interface, text, application

Description automatically generated

|  |
| --- |
| **Partition Count:** 1 to 32  **Message Retention:** No of days to keep message in event hub for processing.  **Capture:** 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. |

🡪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.

**Lab : Detect Fradulant calls in real time so that either notify the customer or shut down service for specific number**

**Identify multiple calls from same identity at nearly same time in but in different geographical location and write those to Blob Storage**

**Use Telcogenerator App**

1. Create credentials for your event hub

Eventhubnamespace🡪Entities🡪 [fraud-calls-evthubs](https://portal.azure.com/#blade/Microsoft_Azure_EventHub/EventHubOverviewBlade/id/%2Fsubscriptions%2F00e19ae2-4419-4975-b266-74bb63299900%2FresourceGroups%2FDP203-MayRG%2Fproviders%2FMicrosoft.EventHub%2Fnamespaces%2Fdssevthubnamespace%2Feventhubs%2Ffraud-calls-evthubs)🡪Settings🡪Shared Access Policy🡪Add🡪

Graphical user interface, application

Description automatically generated🡪Create

Select policy1🡪Copy Connection string-primary key to notepad

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

1. Configure Telcogenerator app

* Open the TelcoGenerator\TelcoGenerator\telcodatagen.exe.config file
* Update the <appSettings> element in the config file with the following details:
  1. Set the value of the *EventHubName* key to the value of the EntityPath in the connection string.
  2. Set the value of the *Microsoft.ServiceBus.ConnectionString* key to the connection string without the EntityPath value. Don't forget to remove the semicolon that precedes the EntityPath value.

Go to Command prompt🡪Change directory to telcoGenerator APP 🡪Run Telcodatagen application using following command

.\telcodatagen.exe 1000 0.2 2

Configure Azure stream analytics using Event Hub and Blob Storage

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

1. **Create Resource**🡪**Analytics**🡪**Stream Analytics Job**🡪

Graphical user interface, text, application

Description automatically generated **🡪Create**

1. **Configure Input**

Job Topology🡪Inputs🡪Add Stream Input🡪Event Hub🡪

Graphical user interface, text

Description automatically generated

**🡪Save**

1. **Configure Output**

* Create Container **fraud-calls** in blob storage
* Job Topology🡪Outputs🡪

Graphical user interface, text, application

Description automatically generated

1. **Job Topology**🡪**Query**🡪

SELECT

    \*

INTO

    [blob-output]

FROM

    [fraudcall-evtInput]

🡪Save Query

**Phone call data Record details:**

|  |  |
| --- | --- |
| CallrecTime: | The timestamp for the call start time. |
| SwitchNum: | The telephone switch used to connect the call. For this example, the switches are strings that represent the country/region of origin (US, China, UK, Germany, or Australia). |
| CallingNum: | The phone number of the caller. |
| CallingIMSI: | The International Mobile Subscriber Identity (IMSI). It's a unique identifier of the caller. |
| CalledNum: | The phone number of the call recipient. |
| CalledIMSI: | International Mobile Subscriber Identity (IMSI). It's a unique identifier of the call recipient. |

**Start Job Options**

There are three options available for jo 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.

**Test Following Queries**

|  |
| --- |
| SELECT     CallRecTime, SwitchNum, CallingIMSI,CallingNumCalledNum  INTO      [blob-output]  FROM      [evt-input] |
| **Count Incoming calls by region**  SELECT     System.Timestamp as WindowEnd, SwitchNum, COUNT(\*) as CallCount  INTO      [blob-output]  FROM      [evt-input]  TIMESTAMP BY CallRecTime  GROUP BY TUMBLINGWINDOW(s, 5), SwitchNum  Note:  Timestamp By : specify which timestamp field in the input stream to use to define the Tumbling window.  System.Timestamp :which returns a timestamp for the end of each window. |
| Detect fraudulent calls ,the calls that originate from the same user but in different locations within 5 seconds of one another  SELECT      System.Timestamp as WindowEnd,  COUNT(\*) as CallCount  INTO      [blob-output]  FROM      [evt-input]  cs1 TIMESTAMP BY CallRecTime      JOIN [evt-input]  cs2 TIMESTAMP BY CallRecTime       ON CS1.CallingIMSI = CS2.CallingIMSI   AND DATEDIFF(ss, CS1, CS2) BETWEEN 1 AND 5   WHERE CS1.SwitchNum != CS2.SwitchNum   GROUP BY TumblingWindow(Duration(second, 1)) |

**Ref:**

[**https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-window-functions**](https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-window-functions)

[**https://docs.microsoft.com/en-us/stream-analytics-query/stream-analytics-query-language-reference**](https://docs.microsoft.com/en-us/stream-analytics-query/stream-analytics-query-language-reference)

**https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-real-time-fraud-detection**

**Configure Job Output in Power BI**

1. Output🡪+Add🡪Power BI🡪

Graphical user interface, application

Description automatically generated🡪Authorize

1. Start Job
2. Start app to generate data : .\telcodatagen.exe 1000 0.2 2
3. Go to PowerBI Workspace🡪Observe the Dataset
4. Create New Dashboard

Graphical user interface, application

Description automatically generated

1. Edit🡪Add tile🡪Select Custome Streaming

Graphical user interface, application

Description automatically generated🡪Next

Graphical user interface, application

Description automatically generated🡪Next🡪

1. Configure Report

Graphical user interface, application

Description automatically generated🡪Add Tile Details🡪Apply

Configure Multiple Outputs:

WITH [AllCalls] AS

(SELECT System.Timestamp as WindowEnd, COUNT(\*) as CallCount

FROM [fraudcall-evtinput] cs1 TIMESTAMP BY CallRecTime

JOIN [fraudcall-evtinput] cs2 TIMESTAMP BY CallRecTime

ON CS1.CallingIMSI = CS2.CallingIMSI

AND DATEDIFF(ss, CS1, CS2) BETWEEN 1 AND 5

WHERE CS1.SwitchNum != CS2.SwitchNum

GROUP BY TumblingWindow(Duration(second, 1)))

SELECT

\*

INTO

[lake-output]

FROM

[AllCalls]

SELECT

\*

INTO

[bi-output]

FROM

[AllCalls]