Microsoft

Azure Stream Analytics

Jack Bender
Industry Architect
US Manufacturing

Unlocking Real-time Insights

Time to Insight is Critical

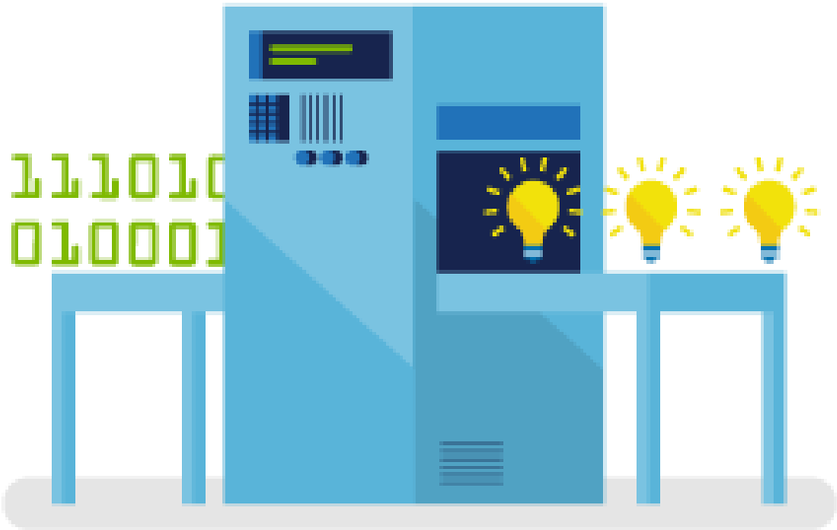
Reducing decision latency can unlock business value

Insights are Perishable

Window of opportunity for insights to be actionable

Ask Questions to Data in Motion

Can’t wait for data to get to rest before running computation



Real-time Stream Processing

Simple Event Processing

- Filter
- Transform
- Enrich
- Split
- Route

Event Stream Processing

- [Simple event processing] +
- Aggregate
- Rules

Complex Event Processing

- [Event Stream Processing] +
- Pattern detection
- Time windows
- Joins & correlations

Scenario Types

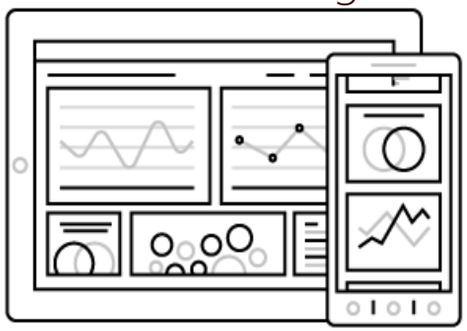
Actions by Human Actors

- “See and seize” insights
- Live visualization
- Alerts and alarms
- Dynamic aggregation

Machine to Machine Interactions

- Data movement with enrichment
- Kick-off workflows for automation

Dashboarding



Automation

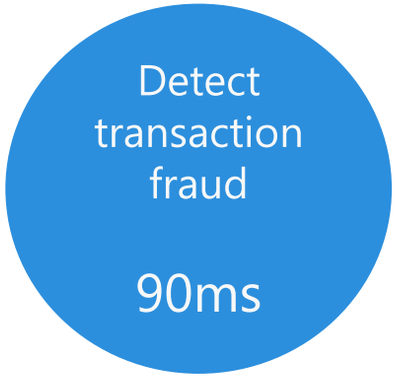
Enriched Data Movement

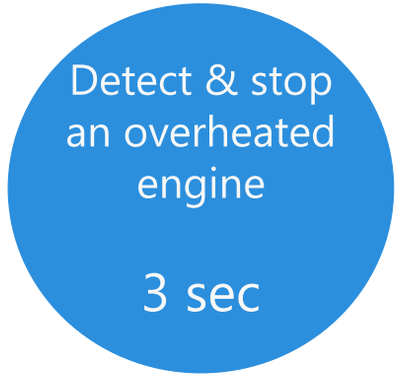
Overview

What is Azure Stream Analytics?

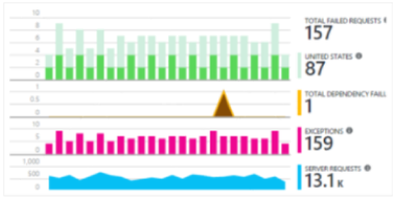
- In-memory Streaming Processing Engine
- Fully Managed: Streaming as a Service!
- Advanced time management made easy

Enables customer to get real-time Insights when time to action is critical





Stream Analytics is relevant across industries

Real-time Fraud Detection 	Streaming ETL 	Predictive Maintenance 	Call Center Analytics 
IT Infrastructure and Network Monitoring 	Customer Behavior Prediction 	Log Analytics 	Real-time Cross Sell Offers 
Fleet monitoring and Connected Cars 	Real-time Patient Monitoring 	Smart Grid 	Real-time Marketing 

ASA key differentiators

Programmer productivity: SQL + Extensibility

Intelligent Edge, and Cloud

Serverless and low TCO

Easy to get started

Enterprise grade SLA (99.9%)

Programmer Productivity:

- Simple SQL language
- Extensibility
  - User-Defined Functions (UDF) and Aggregates (UDA)
    - JavaScript (available)
    - C# (preview on edge), more language to come
- Machine-Learning based Anomaly Detection

Programmer productivity: SQL + Extensibility

Intelligent Edge, and Cloud

Serverless and low TCO

Easy to get started

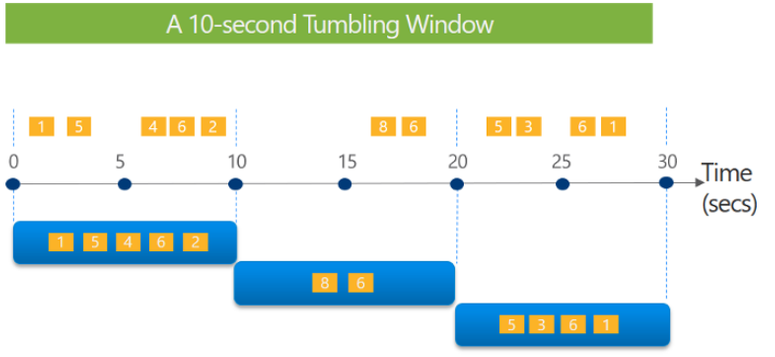
Enterprise grade SLA (99.9%)

POWERFUL ANALYTICS WITH SIMPLE SQL CONSTRUCTS

- SQL language extended with temporal functions
- Ability to join different streams of data, or streams and reference data
- Time-based windows (between 1 ms and 7 days)
  - Using GROUP BY operator for an easy integration with SQL

Example: Tumbling Window

Tell me the count of tweets per time zone every 10 seconds




```
SELECT TimeZone, COUNT(*) AS Count
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY TimeZone, TumblingWindow(second,10)
```

Stream Analytics Query Language (SAQL)

Declarative SQL like language to describe transformations

Filters (“Where”)

Projections (“Select”)
Time-window and property-based aggregates (“Group By”)
Time-shifted joins (specifying time bounds within which the joining events must occur)

and all combinations thereof

Data Manipulation
SELECT
FROM
WHERE
HAVING
GROUP BY
CASE WHEN THEN ELSE
INNER/LEFT OUTER JOIN
UNION
CROSS/OUTER APPLY
CAST INTO
ORDER BY ASC, DSC

Aggregation
SUM
COUNT
AVG
MIN
MAX
STDEV
STDEVP
VAR
VARP
TopOne

Date and Time
DateName
DatePart Day, Month, Year
DateDiff
DateTimeFromParts
DateAdd

Temporal
Lag
IsFirst
Last
CollectTop

Windowing Extensions
TumblingWindow
HoppingWindow
SlidingWindow

Scaling Extensions
WITH
PARTITION BY
OVER

String
Len
Concat
CharIndex
Substring
Lower, Upper
PatIndex

Mathematical
ABS
CEILING
EXP
FLOOR
POWER
SIGN
SQUARE
SQRT

Geospatial (preview)
CreatePoint
CreatePolygon
CreateLineString
ST_DISTANCE
ST_WITHIN
ST_OVERLAPS
ST_INTERSECTS

Differentiators

1,915 lines of code with Apache Storm

```
@ApplicationAnnotation(name="WordCountDemo")
public class Application implements StreamingApplication
{
	protected String fileName =
	"com/datatorrent/demos/wordcount/samplefile.txt";
		private Locality locality = null;

	@Override  public void populateDAG(DAG dag, Configuration
	conf)
	{
		locality = Locality.CONTAINER_LOCAL;
		WordCountInputOperator input =
		dag.addOperator("wordinput", new
		WordCountInputOperator());
		input.setFileName(fileName);
		UniqueCounter<String> wordCount =
		dag.addOperator("count", new
		UniqueCounter<String>());
		dag.addStream("wordinput-count", input.outputPort,
		wordCount.data).setLocality(locality);
		ConsoleOutputOperator consoleOperator =
		dag.addOperator("console", new
		ConsoleOutputOperator());
		dag.addStream("count-console",wordCount.count,
		consoleOperator.input);
	 }
}
```

3 lines of SQL in Azure Stream Analytics

```
SELECT Avg(Purchase), ScoreTollId, Count(*)
FROM GameDataStream
GROUP BY TumblingWindows(5, Minute), Score
```

Intelligent Edge and Cloud

- Same queries running either on Edge or Cloud
- Leveraging IoT Edge for security and deployment
- No equivalent service from our main competitors
- Working on Hybrid jobs

Programmer productivity: SQL + Extensibility

Intelligent Edge, and Cloud

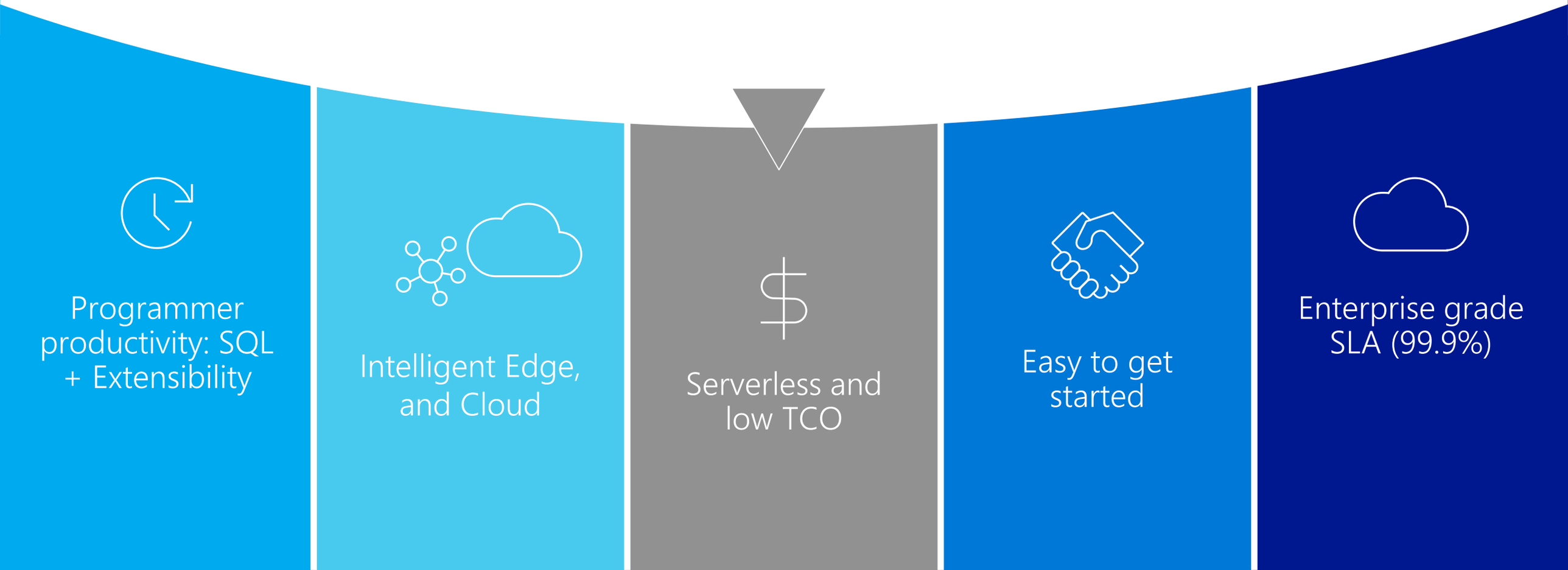
Serverless and low TCO

Easy to get started

Enterprise grade SLA (99.9%)

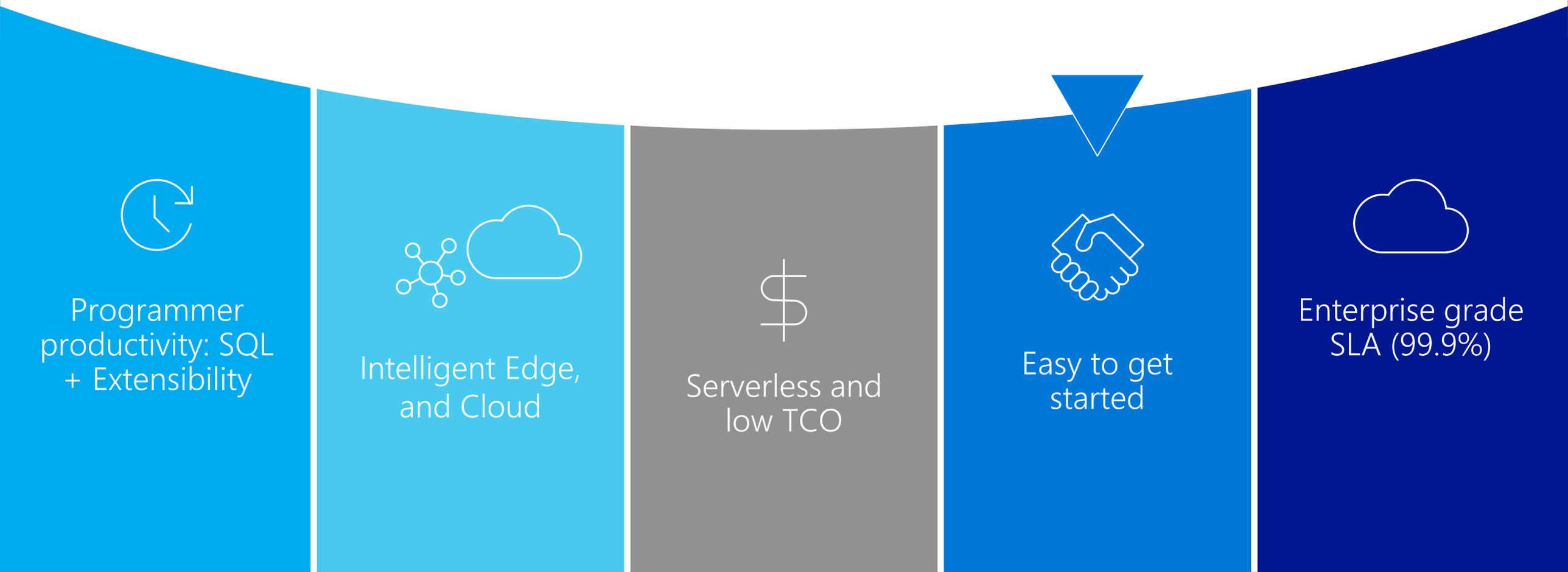
Serverless and low TCO

- Managed job service: no cluster to manage, just run job
- Pay as you go: pay by hour, start, stop, scale when needed – no commitment
- Scale to 120 SUs and beyond

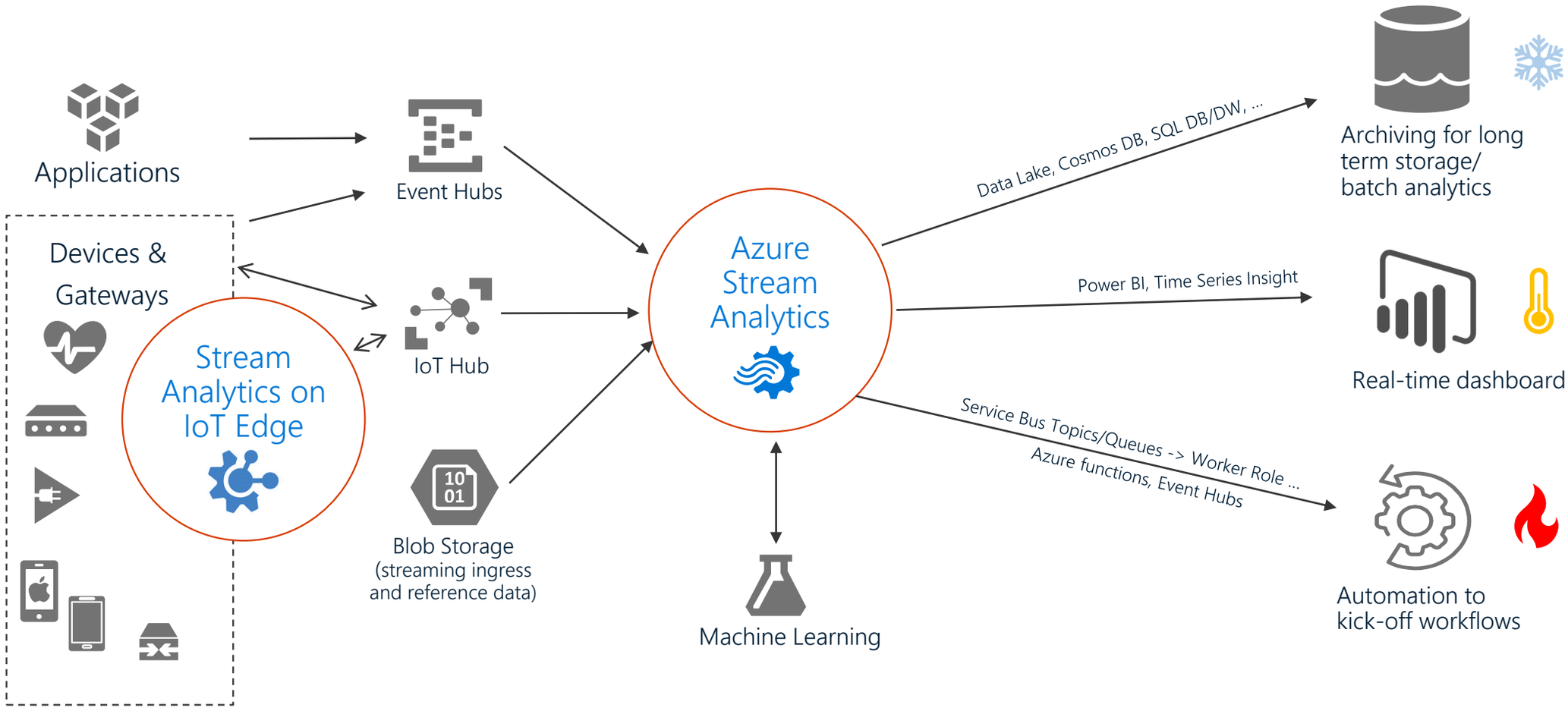


Easy to get started

- Native integration with the rest of Azure Ecosystem
- “5 minutes to solution” without writing any code



ANALYTICS WITH AZURE STREAM ANALYTICS



Event production ⟷ Event Queuing & Stream Ingestion → Stream Analytics → Storage & Batch Analysis → Presentation & Action

Enterprise grade SLA

- Unique SLA in the industry: 99.9% at the minute level
  - Google Dataflow provides 99.5%, AWS Kinesis Analytics doesn't provide SLA
- Working towards 99.99% and helping customers to write pipelines designed to run 24/7

Programmer productivity: SQL + Extensibility

Intelligent Edge, and Cloud

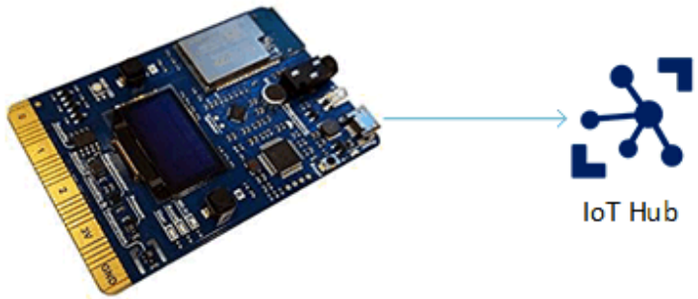
Serverless and low TCO

Easy to get started

Enterprise grade SLA (99.9%)

Components of ASA

Data Streaming from a Device



2/27/2018 5:05:38 PM> Device: [JBAZ3166], Data:
[{"humidity":37.10,"temp":21.50,"pressure":1014.34,"magnetometerX":994,"magnetometerY":613,"magnetometerZ":1659,
"accelerometerX":-12,"accelerometerY":13,"accelerometerZ":997,"gyroscopeX":-
1330,"gyroscopeY":1190,"gyroscopeZ":350}]Properties:
'timestamp': 'Tue Feb 27 22:05:37 2018'

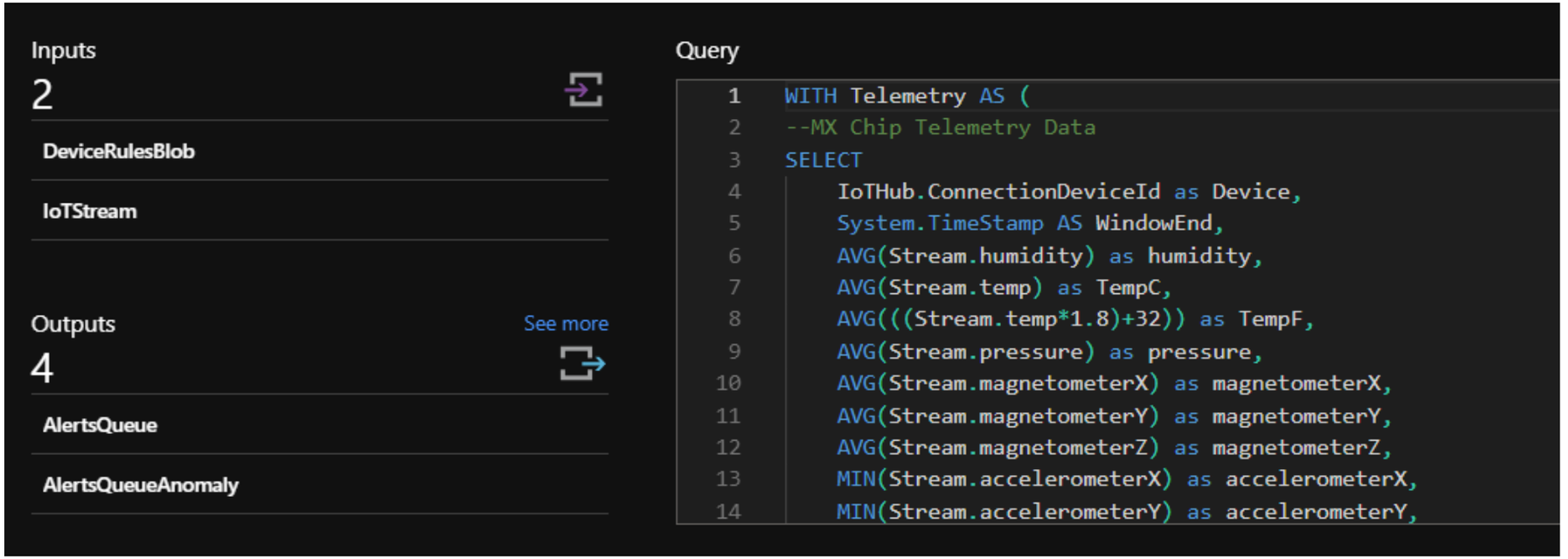
Stream Analytics Job

Job definition includes inputs, a query, and output

Inputs are from where the job reads the data stream

Query runs for perpetuity unless explicitly stopped and transforms the input stream

Output is where the job sends the job results to



Stream Analytics Job Inputs & Outputs

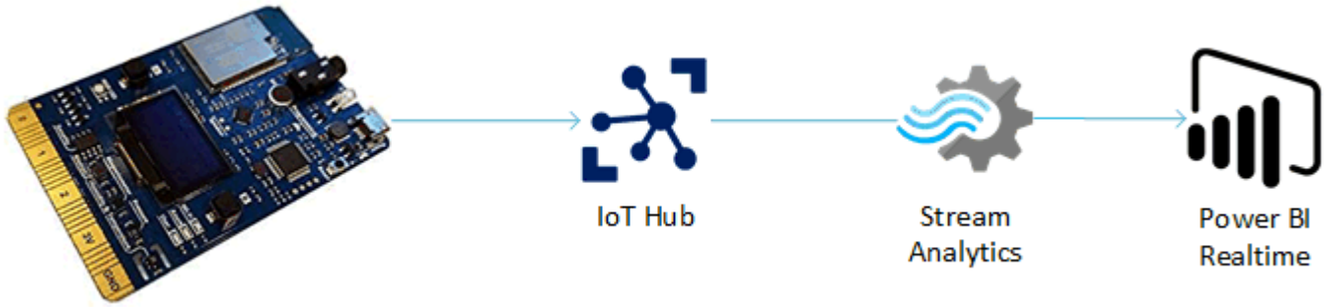
There are two distinct types of Inputs

- **Data Streams**:
  - IoT Hub
  - Event Hub
  - Azure Blob storage
- **Reference data**:
  - Azure Blob storage

Data Outputs Supported

- Azure Data Lake Store
- SQL Database
- Blob storage
- Event Hub
- Power BI
- Table Storage
- Service Bus Queues
- Service Bus Topics
- Azure Cosmos DB
- Azure Functions

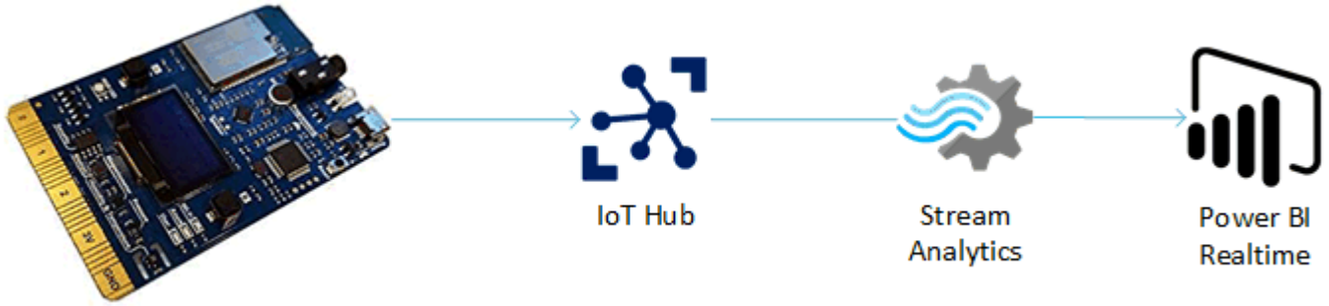
Basic Query to Power BI Realtime




```
WITH Telemetry AS (
--MX Chip Telemetry Data
SELECT
    Stream.*
FROM
    IoTStream Stream
)

SELECT
Telemetry.*
INTO PowerBI
from Telemetry
```

Enhanced Query to Power BI Realtime



- Device ID/Name
- Time
- Fahrenheit

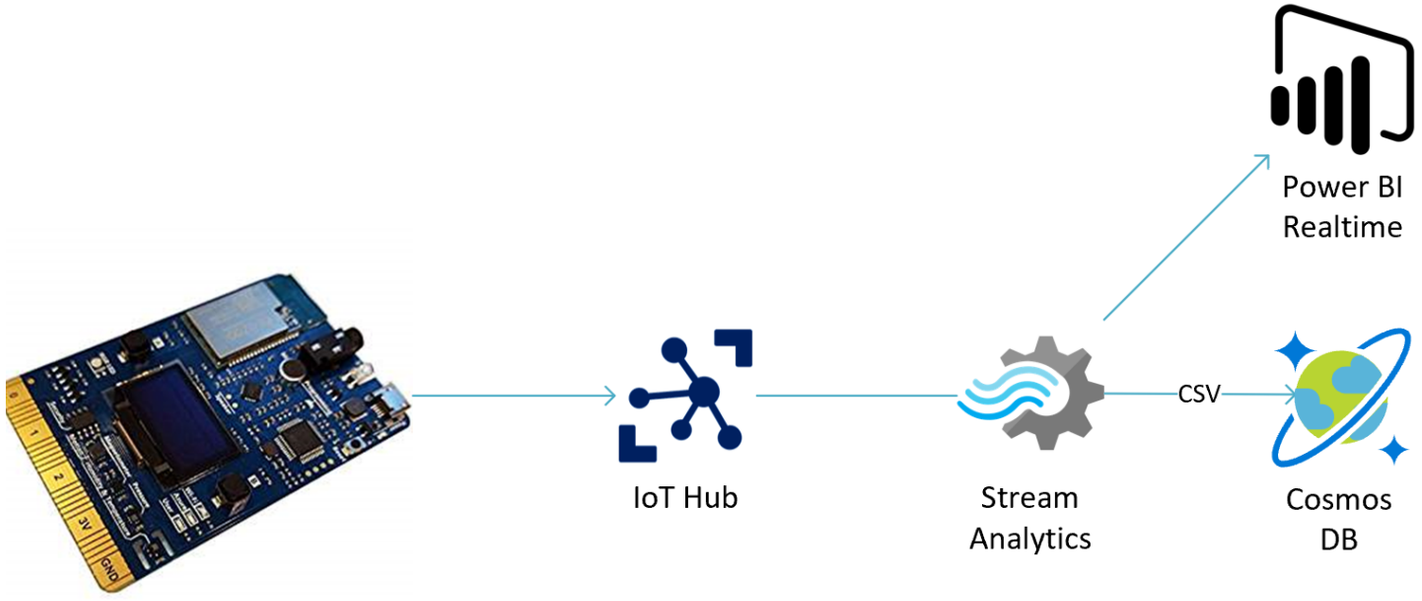
```
2/27/2018 5:05:38 PM> Device: [JBAZ3166], Data:
[{"humidity":37.10,"temp":21.50,"pressure":1014.34,"magnetometerX":994,"magnetome
"accelerometerX":-12,"accelerometerY":13,"accelerometerZ":997,"gyroscopeX":-
1330,"gyroscopeY":1190,"gyroscopeZ":350}]Properties:
'timestamp': 'Tue Feb 27 22:05:37 2018'
```

```
WITH Telemetry AS (
--MX Chip Telemetry Data
SELECT
    IoTHub.ConnectionDeviceId as Device,
    IoTHub.EnqueuedTime as Time,
    Stream.humidity,
    Stream.temp as TempC,
    ((Stream.temp*1.8)+32) as TempF,
    Stream.pressure,
    Stream.magnetometerX,
    Stream.magnetometerY,
    Stream.magnetometerZ,
    Stream.accelerometerX,
    Stream.accelerometerY,
    Stream.accelerometerZ,
    Stream.gyroscopeX,
    Stream.gyroscopeY,
    Stream.gyroscopeZ

FROM
    IoTStream Stream
)

SELECT
Telemetry.*
INTO PowerBI
from Telemetry
```

Data to Cosmos DB & Power BI Realtime

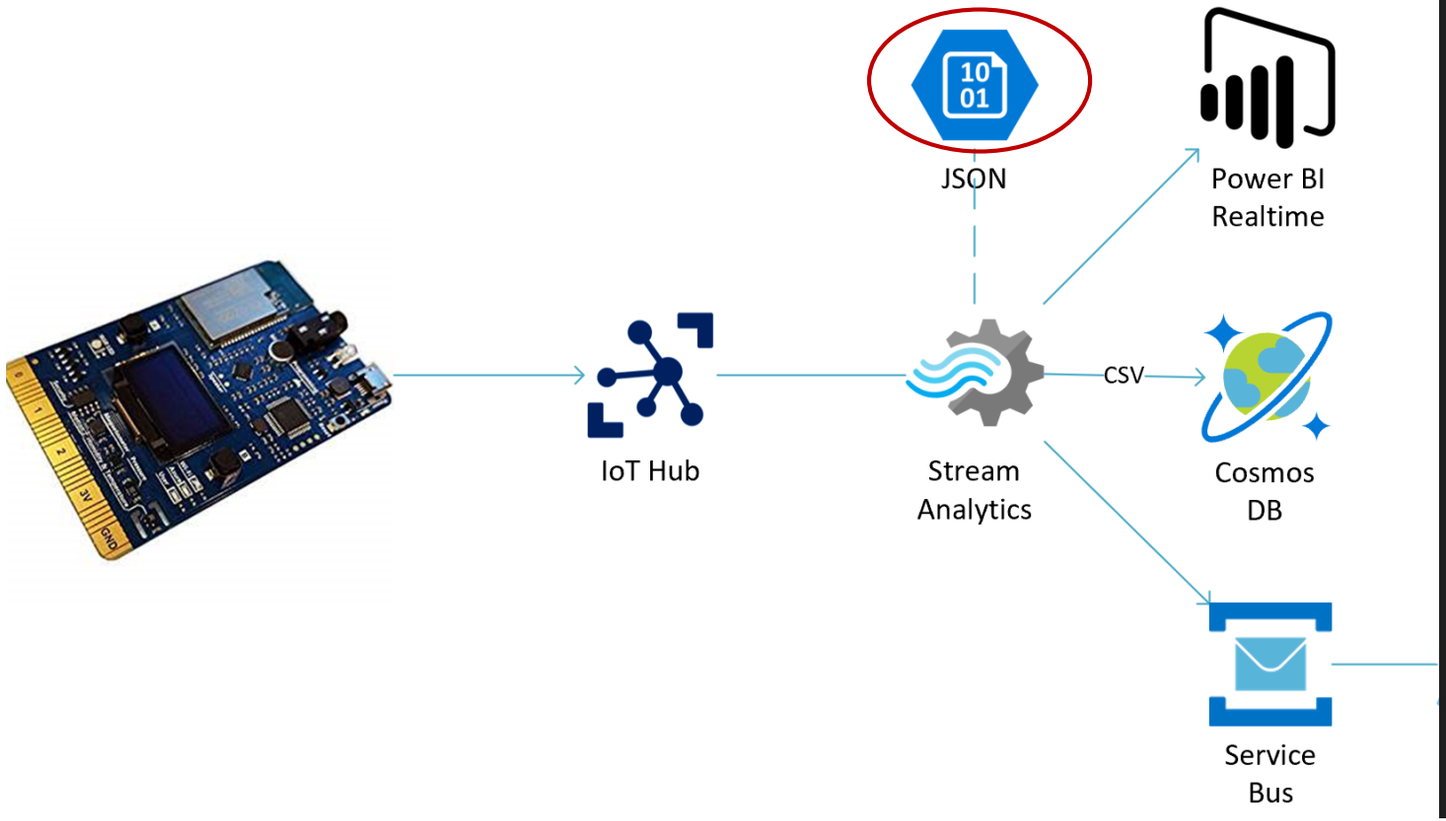



```
24  -- Send all data to Cosmos DB
25  SELECT
26  Telemetry.*
27  INTO CosmosDB
28  from Telemetry
29
30  -- Send some data to Power BI
31  SELECT
32  Telemetry.Device,
33  Telemetry.Time,
34  Telemetry.accelerometerX as 'Accelerometer X',
35  Telemetry.accelerometerY as 'Accelerometer Y',
36  Telemetry.accelerometerZ as 'Accelerometer Z',
37  Telemetry.TempF as Fahrenheit,
38  Telemetry.TempC as Celsius
39
40  INTO PowerBI
41  from Telemetry
```

Cosmos DB, Power BI Realtime and Text Alert

devicerules.json

```
[{"DeviceType":"Accelerometer","AccelerometerX":0,"AccelerometerZ":0,"AccelerometerXRuleOutput":"XAlert","AccelerometerZRuleOutput":"ZAlert"}]
```




```
SELECT
Telemetry.*
INTO CosmosDB
from Telemetry

-- Send some data to Power BI
SELECT
Telemetry.Device,
Telemetry.WindowEnd as Time,
Telemetry.accelerometerX as 'Accelerometer X',
Telemetry.accelerometerY as 'Accelerometer Y',
Telemetry.accelerometerZ as 'Accelerometer Z',
Telemetry.TempF as Fahrenheit,
Telemetry.TempC as Celsius

INTO PowerBI
from Telemetry

-- Send Alerts to Message Queue
SELECT
Telemetry.Device,
Telemetry.WindowEnd as Time,
Telemetry.accelerometerZ as 'Accelerometer Z',
Ref.AccelerometerZ as 'Z Threshold',
Ref.AccelerometerZRuleOutput
INTO AlertsQueue

from Telemetry
JOIN DeviceRulesBlob Ref ON Ref.DeviceType = 'Accelerometer'
WHERE Telemetry.accelerometerZ < Ref.AccelerometerZ
```

Time Functions

Events and Time

Every event that flows through the system has a timestamp

ASA supports:

- Arrival Time - Event timestamps based on arrival time (input adapter clock, e.g., Event Hubs)
- App Time - Event timestamps based on a timestamp field in the actual event tuple

User can pick up App Time from the payload

```
SELECT * FROM EntryStream TIMESTAMP BY EntryTime
```

System can assign timestamps automatically based on the event arrival time

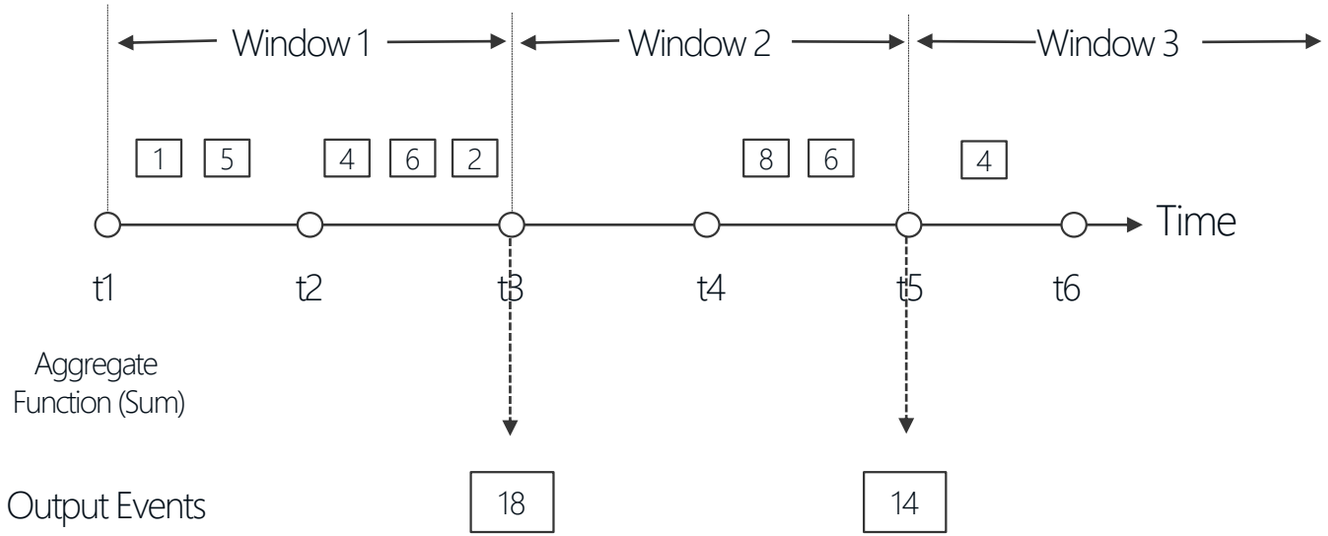
```
SELECT * FROM EntryStream
```

Windowing Concepts

Output at the end of each window

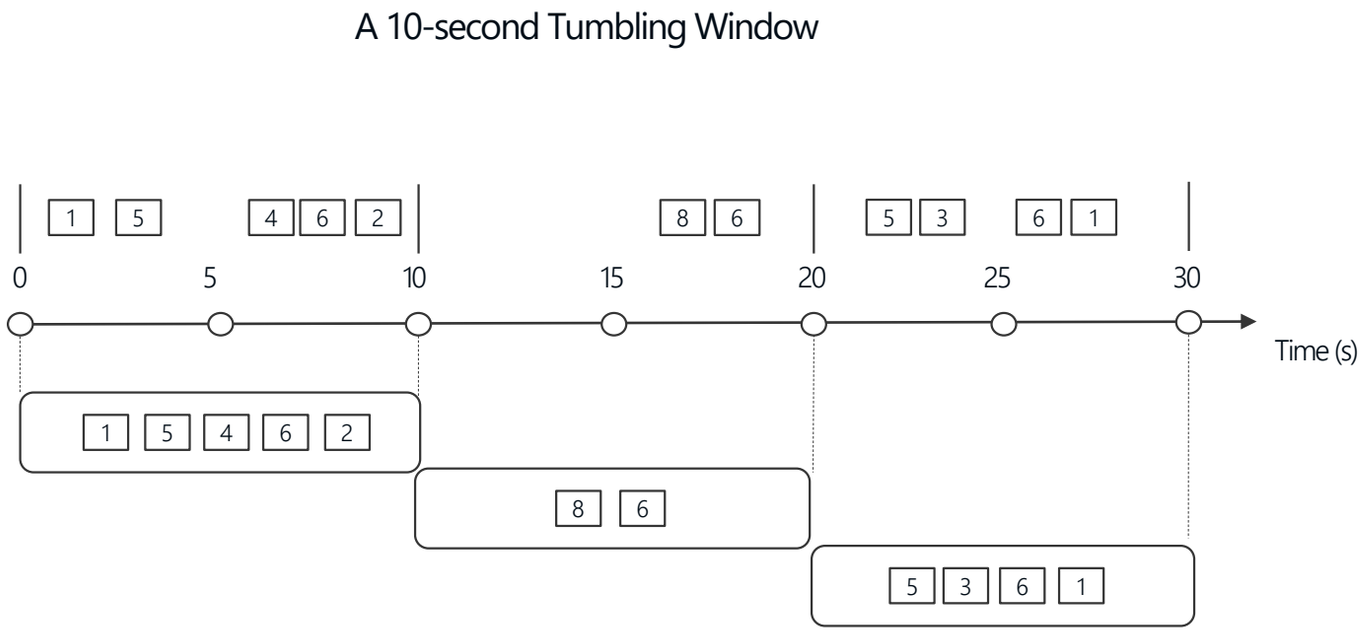
Windows are fixed length

Used in a GROUP BY clause



Tumbling Windows

Every 10 seconds give me the count of vehicles entering each toll booth over the last 10 seconds

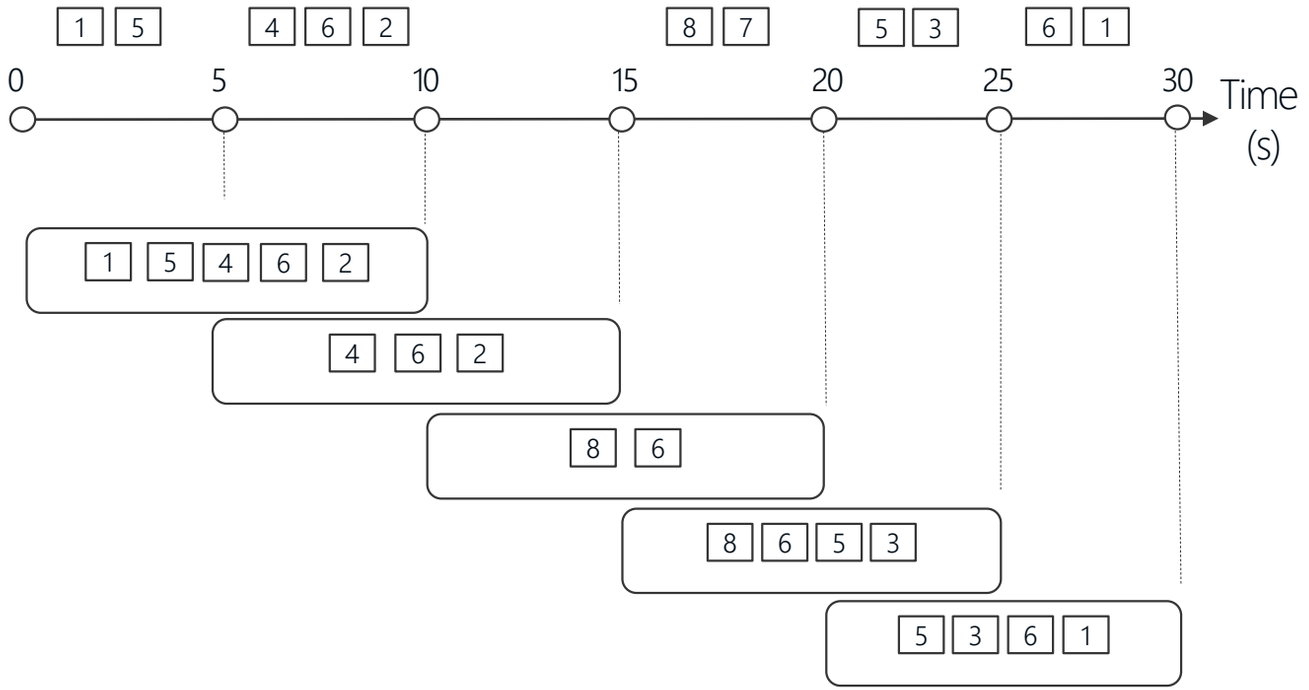



```
SELECT TollId, Count(*)
FROM EntryStream TIMESTAMP BY EntryTime
GROUP BY TollId, TumblingWindow(second, 10)
```

Hopping Windows

Every 5 seconds give me the count of vehicles entering each toll booth over the last 10 seconds

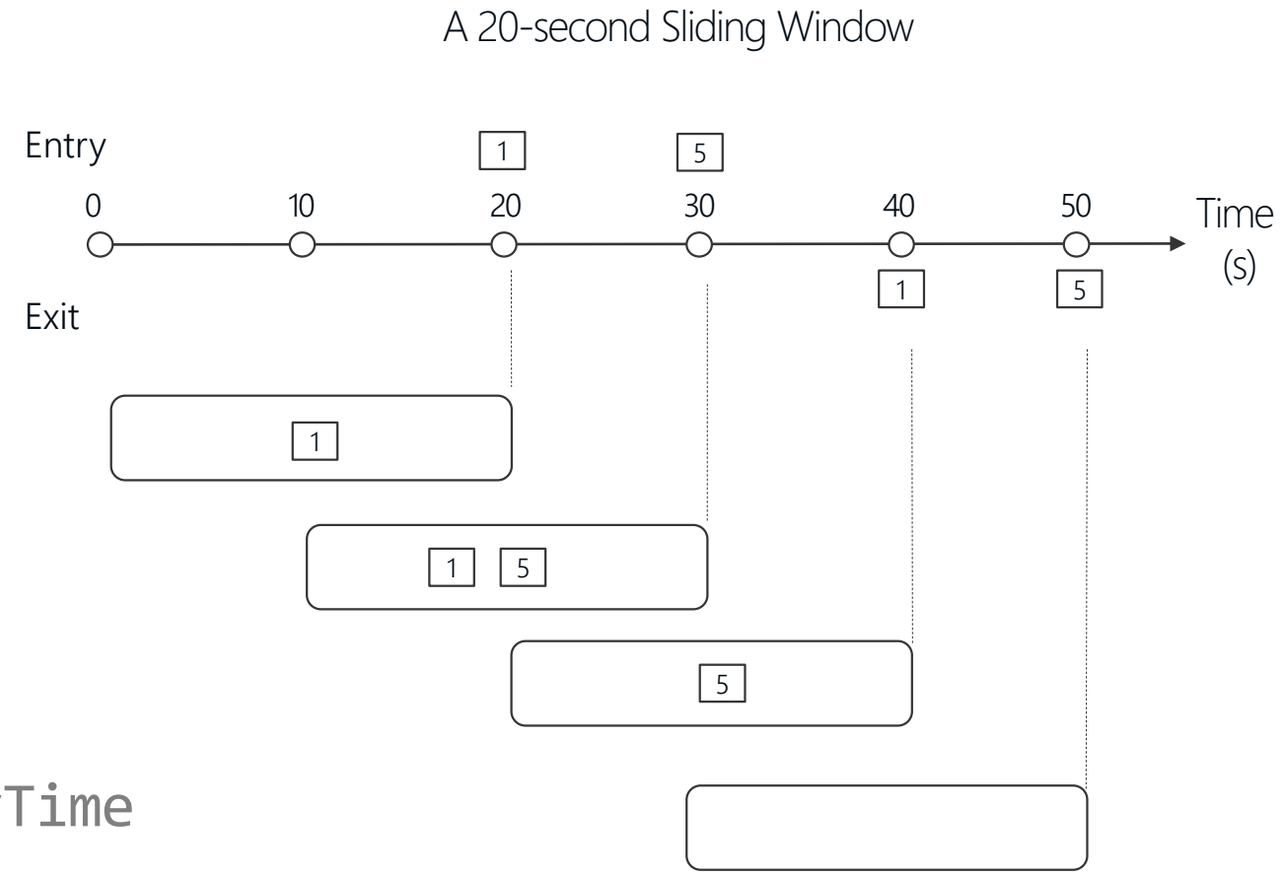
A 10 second Hopping Window with a 5 second hop




```
SELECT TollId, Count(*)
FROM EntryStream TIMESTAMP BY EntryTime
GROUP BY TollId, HoppingWindow(second, 10, 5)
```

Sliding Windows

Find all toll booths that have served more than 10 vehicles in the last 20 seconds




```
SELECT TollId, Count(*)
FROM EntryStream TIMESTAMP BY EntryTime
GROUP BY TollId, SlidingWindow(second, 20)
HAVING Count(*) > 10
```

An output is generated whenever an event either enters/leaves the system

Machine Learning Integration

in public preview

Azure Machine Learning Callouts

Perform real-time scoring on streaming data

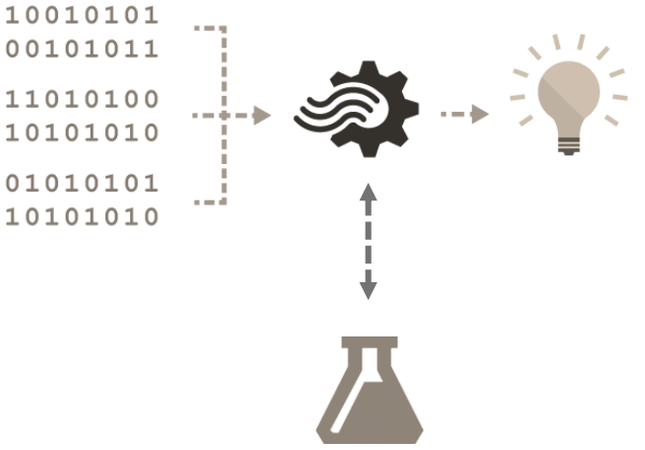
Anomaly Detection and Sentiment Analysis are common use cases

Function calls from the query

Azure ML can publish web endpoints for operationalized ML models

Azure Stream Analytics binds custom function names to such web endpoints

```
SELECT text, sentiment(text) AS score
FROM myStream
```



Azure Machine Learning Callouts

in public preview

```
WITH AlertData AS (
-- Web Simulator Devices
 SELECT
   Stream.EventToken as ID,
   Stream.DeviceID,
   Stream.Temperature AS reading,
   Stream.EventEnqueuedUtcTime AS [time]

from IoTStream  Stream

WHERE
   (Stream.EventToken IS NOT NULL) and (Stre

UNION

-- MX Chip
SELECT
    Stream.messageId AS ID,
    Stream.DeviceID,
    Stream.Temperature AS reading,
    Stream.EventEnqueuedUtcTime AS [time]

FROM IoTStream Stream

WHERE
     (Stream.messageId IS NOT NULL) and (Stre

UNION

-- Particle Device
SELECT
     Stream.published_at as ID,
     Stream.device_id AS DeviceID,
     CAST(Stream.data as float) AS reading,
     Stream.EventEnqueuedUtcTime AS [time]

FROM IoTStream Stream
WHERE
     CAST(Stream.data as float) > 0 and Stream.Event='temperature'
),
```

```
AmlStep AS (
  SELECT
    ID,
    DeviceID,
    reading,
    time,
    aml(ID, DeviceID, reading, time) as result
  from AlertData

)

SELECT
    result.ID,
    result.DeviceID,
    'Temperature' AS ReadingType,
    result.reading,
    CAST(result.time AS datetime) as time,
    result.[abn]

--INTO testoutput
INTO AlertsQueue
FROM AmlStep
WHERE CAST(result.[abn] AS bigint)=1
AND (LAG(result.DeviceID) OVER (PARTITION BY result.DeviceID, result.reading LIMIT DURATION(minute, 1)) IS NULL)
```

Demo

in private preview

SpikeAndDipScores in Azure Stream Analytics

```
WITH Telemetry AS (
--MX Chip Telemetry Data
SELECT
    IoTHub.ConnectionDeviceId as Device,
    System.TimeStamp AS WindowEnd,
    AVG(Stream.humidity) as humidity,
    AVG(Stream.temp) as TempC,
    AVG(((Stream.temp*1.8)+32)) as TempF,
    AVG(Stream.pressure) as pressure,
    AVG(Stream.magnetometerX) as magnetometerX,
    AVG(Stream.magnetometerY) as magnetometerY,
    AVG(Stream.magnetometerZ) as magnetometerZ,
    MIN(Stream.accelerometerX) as accelerometerX,
    MIN(Stream.accelerometerY) as accelerometerY,
    MIN(Stream.accelerometerZ) as accelerometerZ,
    AVG(Stream.gyroscopeX) as gyroscopeX,
    AVG(Stream.gyroscopeY) as gyroscopeY,
    AVG(Stream.gyroscopeZ) as gyroscopeZ

FROM
    IoTStream Stream TIMESTAMP BY IoTHub.EnqueuedTime
GROUP BY IoTHub.ConnectionDeviceId, TumblingWindow(second, 15)

),

-- Anomaly Detection Spike-Dip
AnomalyDetectionStep AS (
SELECT
    *,
    system.timestamp as anomalyDetectionStepTimestamp,
    AnomalyDetection_SpikeAndDip(accelerometerZ, 99, 20, 'spikesanddips') OVER(LIMIT DURATION(mi, 10)) as SpikeAndDipScores
FROM Telemetry
)
```

It is recommended to have at least 50 events in each window for best results.

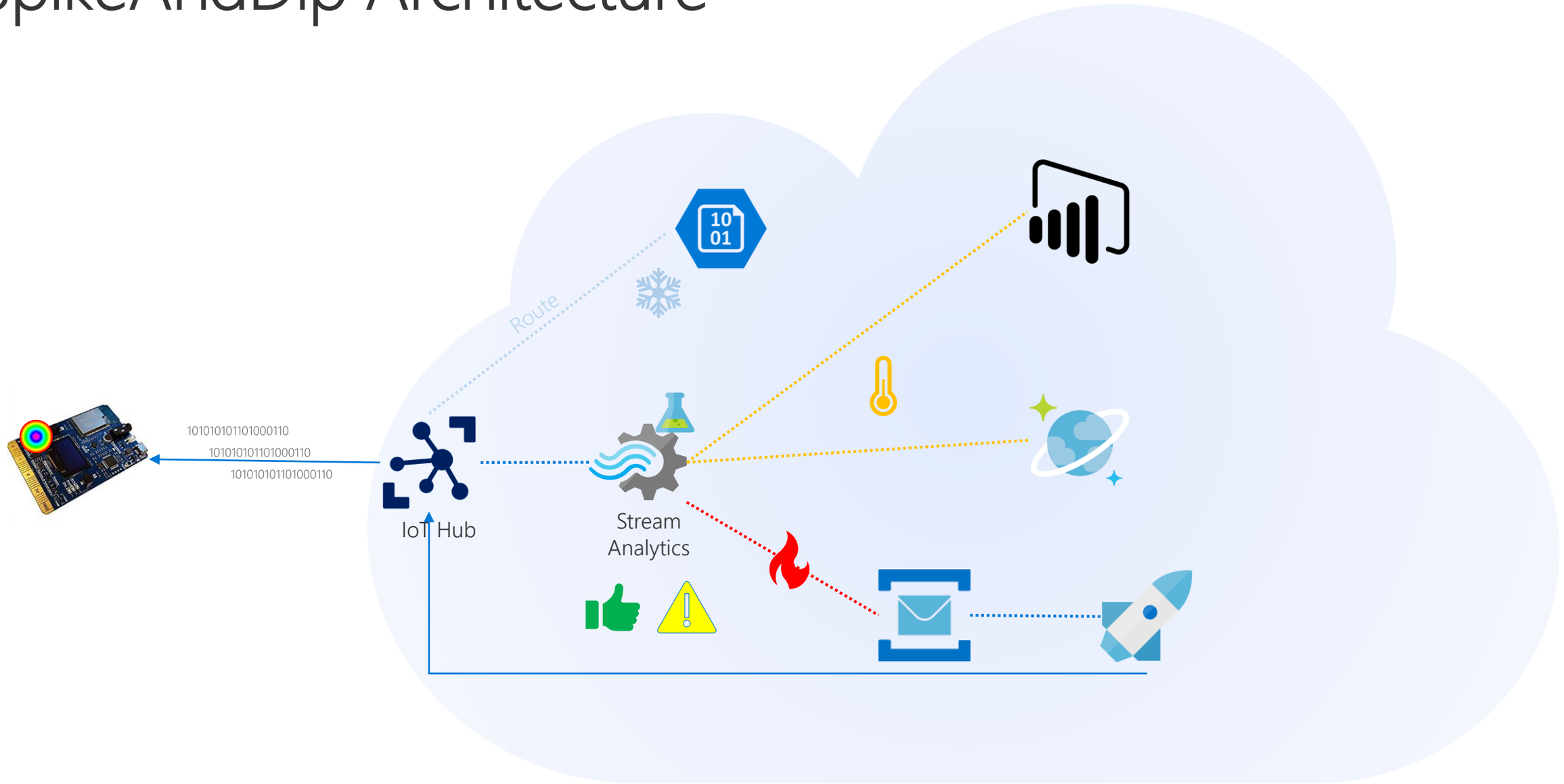
in private preview

SpikeAndDipScores in Azure Stream Analytics

```
58  -- Send some data to Power BI
59  SELECT
60      Device,
61      WindowEnd as Time,
62      accelerometerX as 'Accelerometer X',
63      accelerometerY as 'Accelerometer Y',
64      accelerometerZ as 'Accelerometer Z',
65      TempF as Fahrenheit,
66      TempC as Celsius,
67      CAST(GetRecordPropertyValue(SpikeAndDipScores, 'Score') as float) as SpikeAndDipScore,
68      CAST(GetRecordPropertyValue(SpikeAndDipScores, 'IsAnomaly') as bigint) as IsSpikeAndDipAnomaly
69
70  INTO PowerBI
71  from AnomalyDetectionStep
72
73  -- Send Alerts to Message Queue
74
75  SELECT
76      Device,
77      WindowEnd as Time,
78      'Accelerometer Z' AS ReadingType,
79      accelerometerZ as reading,
80      CAST(GetRecordPropertyValue(SpikeAndDipScores, 'IsAnomaly') as bigint) as abn
81
82  INTO AlertsQueueAnomaly
83  FROM AnomalyDetectionStep
84  WHERE CAST(GetRecordPropertyValue(SpikeAndDipScores, 'IsAnomaly') as float) = 1
```

It is recommended to have at least 50 events in each window for best results.

SpikeAndDip Architecture



JavaScript user-defined functions

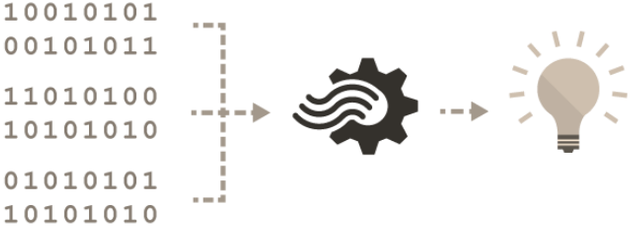
JavaScript user-defined functions

Scenarios where you might find JavaScript user-defined functions useful:

- Parsing and manipulating strings that have regular expression functions, for example, **Regexp_Replace()** and **Regexp_Extract()**
- Decoding and encoding data, for example, binary-to-hex conversion
- Performing mathematic computations with JavaScript **Math** functions
- Performing array operations like sort, join, find, and fill

Here are some things that you cannot do with a JavaScript user-defined function in Stream Analytics:

- Call out external REST endpoints, for example, performing reverse IP lookup or pulling reference data from an external source
- Perform custom event format serialization or deserialization on inputs/outputs
- Create custom aggregates



JavaScript user-defined functions example

```
{
 "devicename": "bigmachine1",
 "datetime": "2018-03-01T08:09:09.0470575-05:00",
 "parameter": "pressure",
 "value": 12
}
{
 "devicename": "bigmachine1",
 "datetime": "2018-03-01T08:09:09.0470575-05:00",
 "parameter": "gigathings",
 "value": 4
}
{
 "devicename": "bigmachine1",
 "datetime": "2018-03-01T08:09:09.0470575-05:00",
 "parameter": "quarks",
 "value": 95
}
{
 "devicename": "bigmachine1",
 "datetime": "2018-03-01T08:09:09.0470575-05:00",
 "parameter": "doodads",
 "value": 12
}
```

The JavaScript UDF function takes an array of json objects and flattens it:

```
function main(arr) {
  //initialize the result object as a blank object
  var result = {};
  if (Object.prototype.toString.call(arr) == "[object Array]") {
      //iterate through all the objects in the array:
      for (i = 0; i < arr.length; i++) {
        //get the value of the datetime parameter and add it to the result array
        //if we haven't already
        if (result["datetime"] == null) {
           result["datetime"] = arr[i].datetime;
        }

        //get the value of the devicename and add it to the result array
        //if we haven't already
        if (result["devicename"] == null) {
          result["devicename"] = arr[i].devicename;
        }

        //check to see if we have added a parameter/value pair to the results array
        //and add it to the results array if it isn't in there
        if(result[arr[i].parameter]==null) {
             result[arr[i].parameter]=arr[i].value;
        }

      }
  }
  //pass the array back to the query
  return result;
}
```

Provided by Rob Klause 

JavaScript user-defined functions example

The ASA query

```
WITH EventCollection AS
(
  SELECT collect() AS allEvents
  FROM iotevents TIMESTAMP BY datetime
  GROUP BY System.Timestamp,devicename
),
PivotRecord AS
(
  SELECT udf.getValues(allEvents) AS record
  FROM EventCollection
)
SELECT record.*
INTO iotoutput
FROM PivotRecord
```

The resulting file is a CSV blob:

```
datetime,devicename,pressure,quarks,gigathings,doodads
2018-03-01T05:57:42.5690687Z,bigmachine1,11,81,7,50
2018-03-01T05:58:42.7074603Z,bigmachine1,5,61,9,27
2018-03-01T05:59:42.8404016Z,bigmachine1,3,32,9,9
2018-03-01T06:00:42.9714928Z,bigmachine1,11,58,5,3
2018-03-01T06:01:43.1051537Z,bigmachine1,12,30,3,9
2018-03-01T06:02:43.2765973Z,bigmachine1,8,86,12,27
2018-03-01T06:03:43.4033917Z,bigmachine1,6,83,19,41
```

Provided by Rob Klause 

Edge Analytics

Edge Analytics

Local Execution

Stream analytics runs on 'edge devices'

Unlock the Value of Untapped data

Only ~5% of data in industrial processes is sent to the cloud

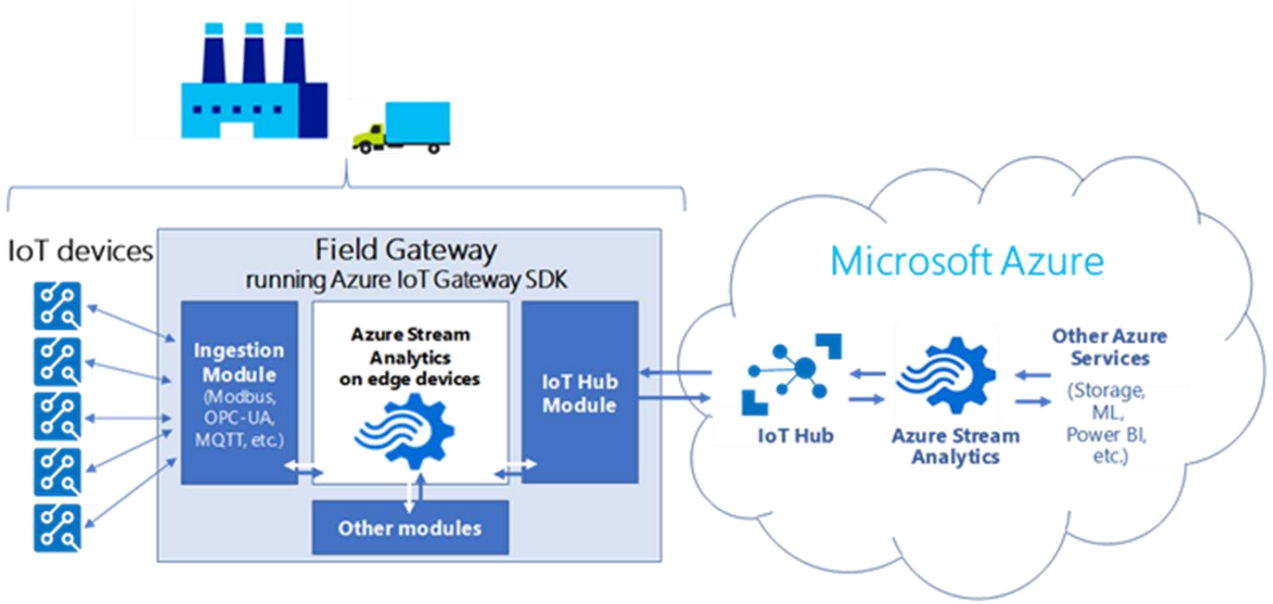
Deploy intelligence near the data to unlock the full value of data

Seamless development and operations

Stream analytics jobs run in the cloud and on edge devices

Intelligent actions

Deploy situational awareness, custom code, ML models on the edge



Edge Analytics Scenarios

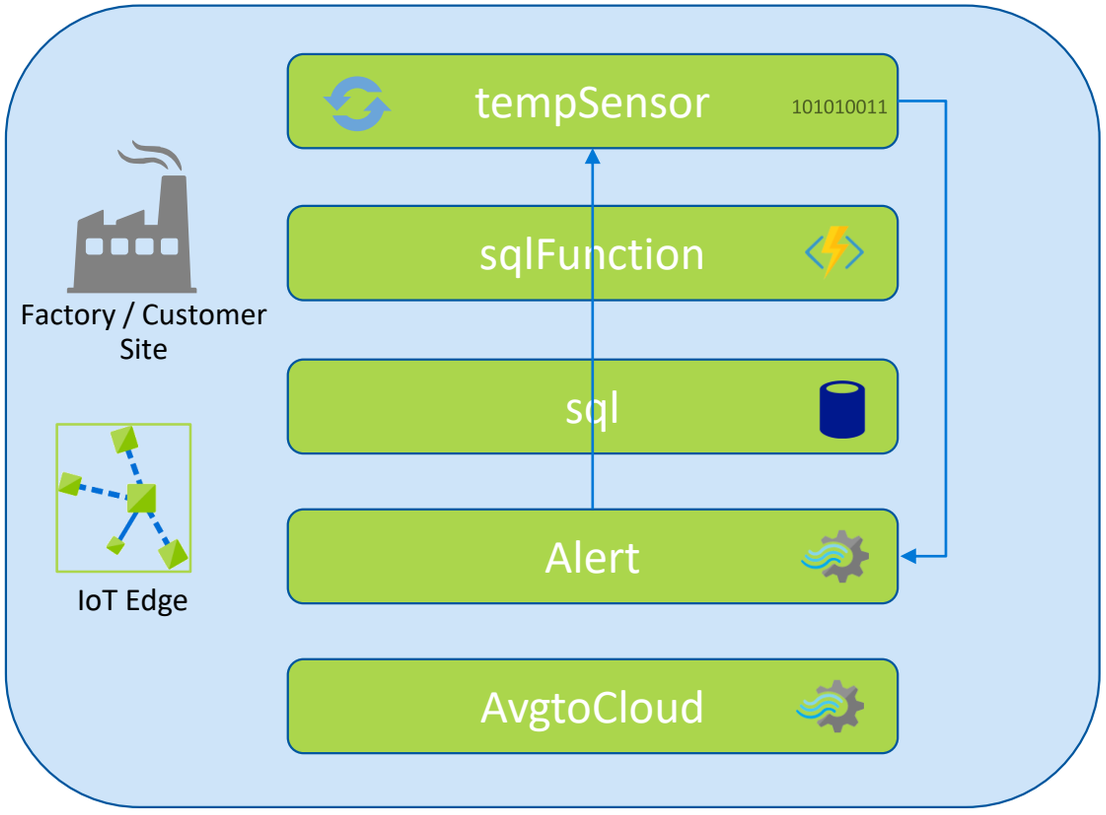
Low Latency

Resiliency

Efficient Use of Bandwidth

Compliance

Management at IoT-Scale



Inputs

1

temperature

Outputs

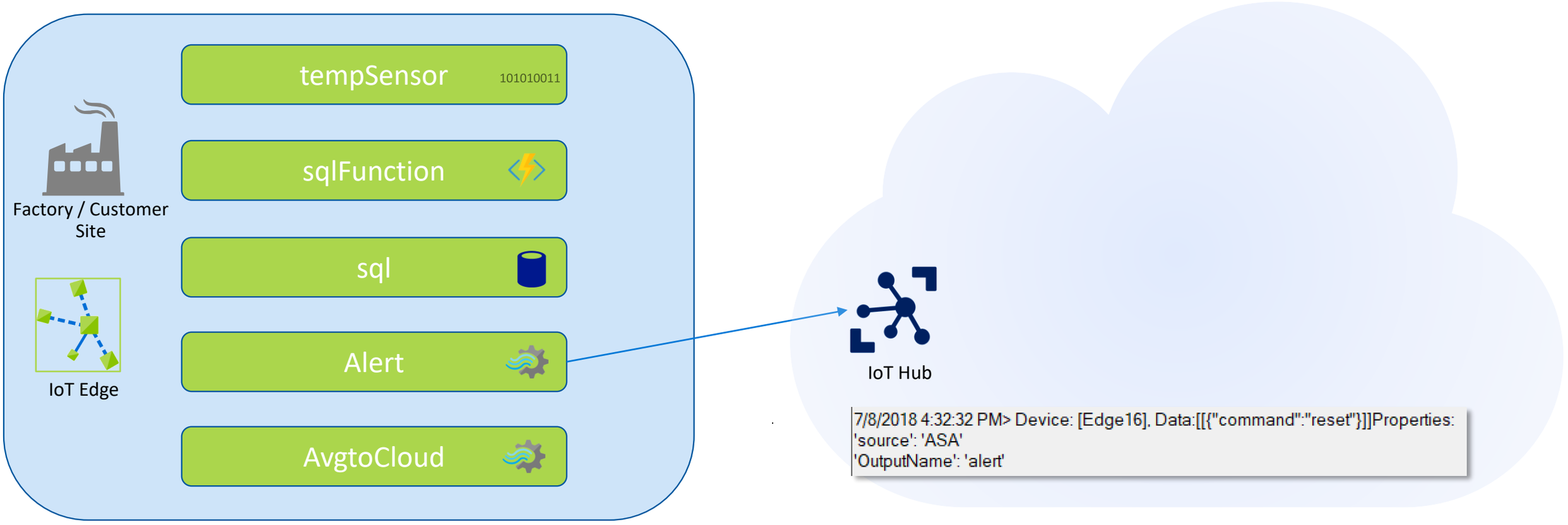
1

alert

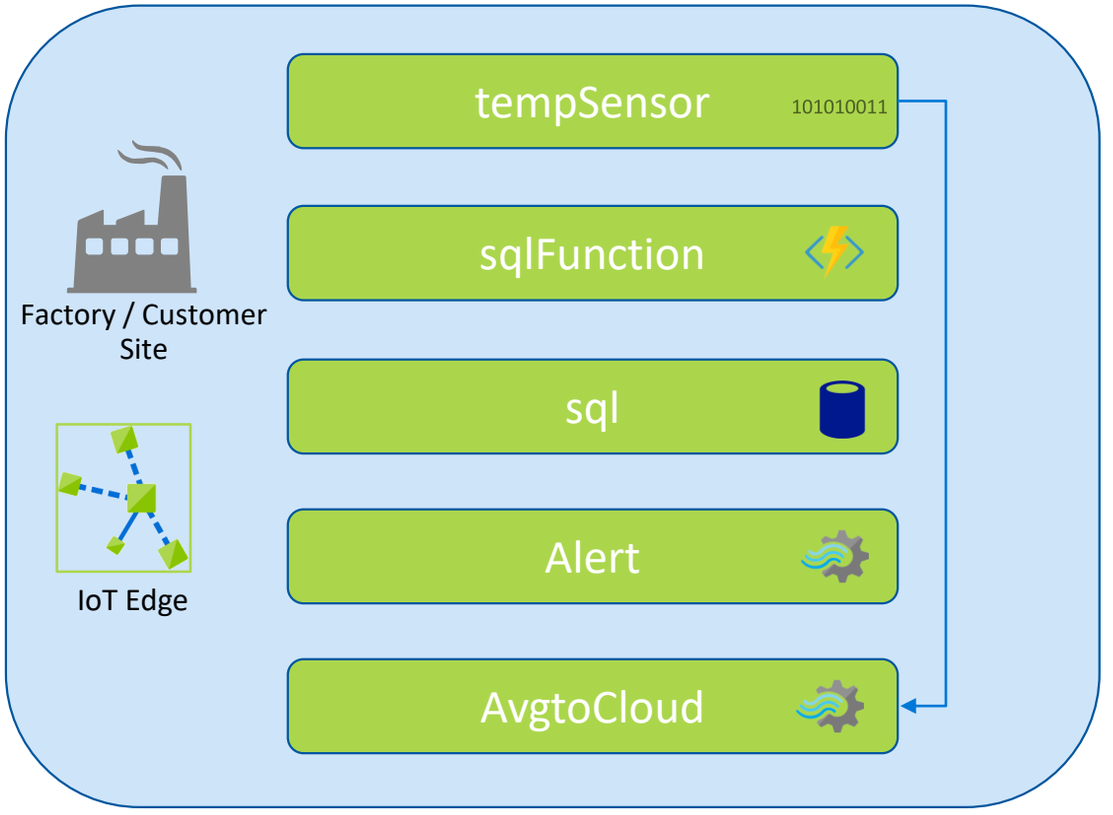
Query

```
SELECT
    'reset' AS command
INTO
   alert
FROM
   temperature TIMESTAMP BY timeCreated
   GROUP BY TumblingWindow(second,30)
   HAVING Avg(machine.temperature) > 70
```

```
{
  "routes": {
              "TelemetryTosqlFunction": "FROM /messages/modules/tempSensor/outputs/temperatureOutput INTO BrokeredEndpoint(\"/modules/sqlFunction/inputs/input1\")",
              "TelemetryToAsa": "FROM /messages/modules/tempSensor/* INTO BrokeredEndpoint(\"/modules/Alert/inputs/temperature\")",
              "AlertsToReset": "FROM /messages/modules/Alert/* INTO BrokeredEndpoint(\"/modules/tempSensor/inputs/control\")",
              "AlertsToCloud": "FROM /messages/modules/Alert/* INTO $upstream",
              "TelemetryToAsaAvg": "FROM /messages/modules/tempSensor/* INTO BrokeredEndpoint(\"/modules/AvgtoCloud/inputs/EdgeStream\")",
              "AvgToCloud": "FROM /messages/modules/AvgtoCloud/* INTO $upstream"
  }
}
```




```
{
  "routes": {
              "TelemetryTosqlFunction": "FROM /messages/modules/tempSensor/outputs/temperatureOutput INTO BrokeredEndpoint(\"/modules/sqlFunction/inputs/input1\")",
              "TelemetryToAsa": "FROM /messages/modules/tempSensor/* INTO BrokeredEndpoint(\"/modules/Alert/inputs/temperature\")",
              "AlertsToReset": "FROM /messages/modules/Alert/* INTO BrokeredEndpoint(\"/modules/tempSensor/inputs/control\")",
              "AlertsToCloud": "FROM /messages/modules/Alert/* INTO $upstream",
              "TelemetryToAsaAvg": "FROM /messages/modules/tempSensor/* INTO BrokeredEndpoint(\"/modules/AvgtoCloud/inputs/EdgeStream\")",
              "AvgToCloud": "FROM /messages/modules/AvgtoCloud/* INTO $upstream"
  }
}
```



Inputs
1
EdgeStream

Outputs
1
Hub

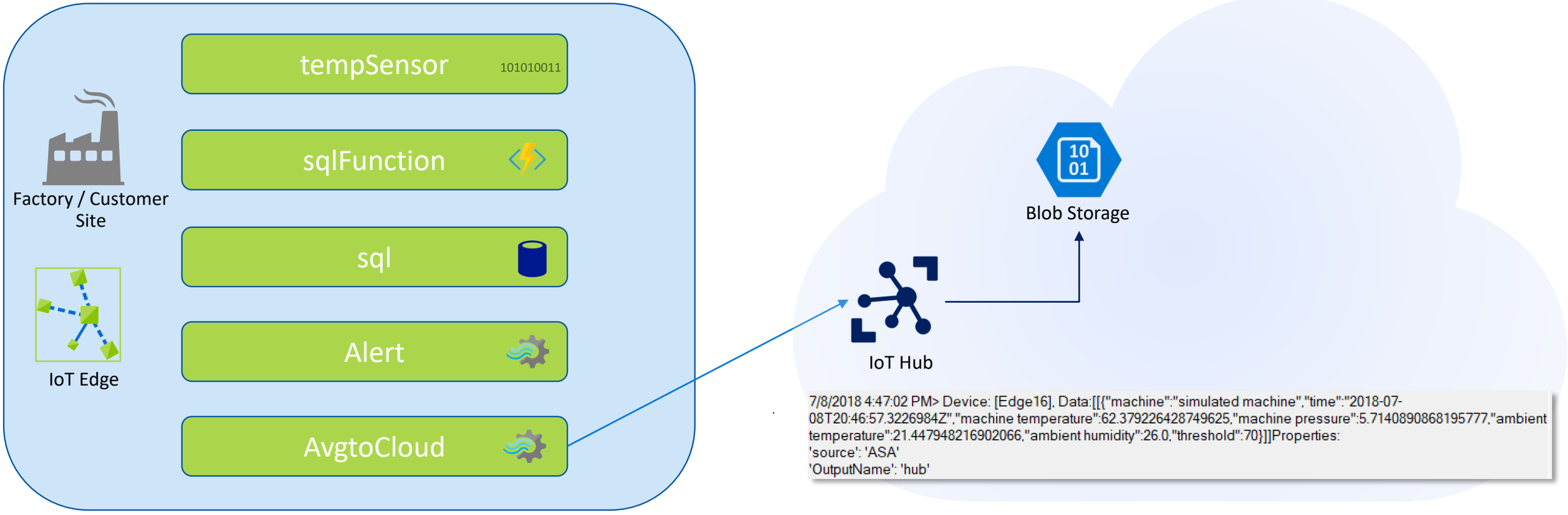
```
WITH Telemetry AS (

SELECT
    'simulated machine' as machine,
    MAX(timeCreated) as time,
    MAX(machine.temperature) as 'machine temperature',
    MAX(machine.pressure) as 'machine pressure',
    MAX(ambient.temperature) as 'ambient temperature',
    MAX(ambient.humidity) as 'ambient humidity',
    70 as threshold

FROM
    EdgeStream TIMESTAMP BY timeCreated
    WHERE machine.temperature IS NOT NULL
    GROUP BY TumblingWindow(second,30)
)

SELECT
Tel.*
INTO Hub
from Telemetry Tel
```

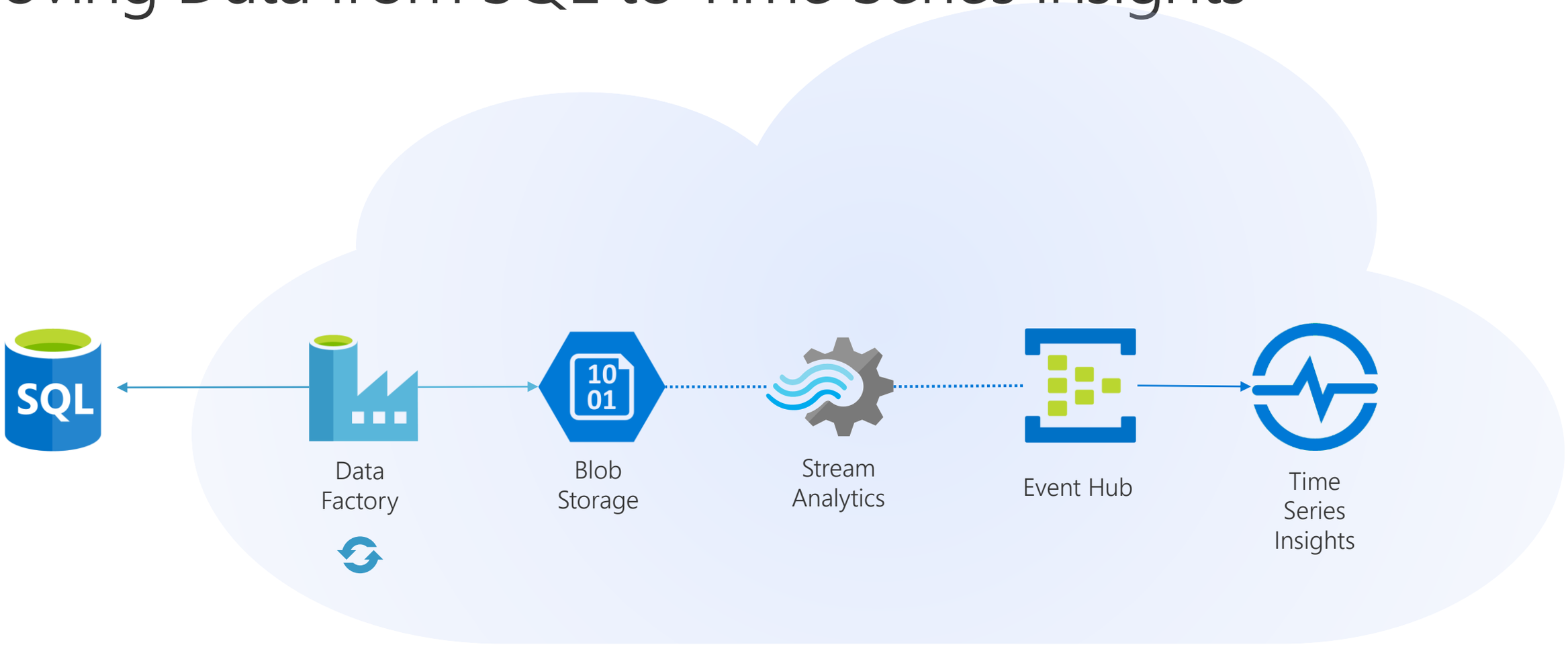
```
{
  "routes": {
              "TelemetryTosqlFunction": "FROM /messages/modules/tempSensor/outputs/temperatureOutput INTO BrokeredEndpoint(\"/modules/sqlFunction/inputs/input1\")",
              "TelemetryToAsa": "FROM /messages/modules/tempSensor/* INTO BrokeredEndpoint(\"/modules/Alert/inputs/temperature\")",
              "AlertsToReset": "FROM /messages/modules/Alert/* INTO BrokeredEndpoint(\"/modules/tempSensor/inputs/control\")",
              "AlertsToCloud": "FROM /messages/modules/Alert/* INTO $upstream",
              "TelemetryToAsaAvg": "FROM /messages/modules/tempSensor/* INTO BrokeredEndpoint(\"/modules/AvgtoCloud/inputs/EdgeStream\")",
              "AvgToCloud": "FROM /messages/modules/AvgtoCloud/* INTO $upstream"
  }
}
```




```
{
  "routes": {
              "TelemetryTosqlFunction": "FROM /messages/modules/tempSensor/outputs/temperatureOutput INTO BrokeredEndpoint(\"/modules/sqlFunction/inputs/input1\")",
              "TelemetryToAsa": "FROM /messages/modules/tempSensor/* INTO BrokeredEndpoint(\"/modules/Alert/inputs/temperature\")",
              "AlertsToReset": "FROM /messages/modules/Alert/* INTO BrokeredEndpoint(\"/modules/tempSensor/inputs/control\")",
              "AlertsToCloud": "FROM /messages/modules/Alert/* INTO $upstream",
              "TelemetryToAsaAvg": "FROM /messages/modules/tempSensor/* INTO BrokeredEndpoint(\"/modules/AvgtoCloud/inputs/EdgeStream\")",
              "AvgToCloud": "FROM /messages/modules/AvgtoCloud/* INTO $upstream"
  }
}
```

Demo – Data Factory ->TSI

Moving Data from SQL to Time Series Insights



New Features

Feature availability details

Feature	Status	Remarks
SQL Parallel Write	GA	WW rollout in 1 week
Blob O/P partitioning by custom date-time	Public preview	WW rollout in 1 week
C# UDF on IoT Edge	Public preview	Available now
Live testing in Visual Studio	Public preview	Available now
User defined custom repartition count	Public preview	Available now
New built-in ML models for A/D – Edge and Cloud	Private Preview	Access granted upon sign-up
Custom de-serializers on IoT Edge	Private Preview	Access granted upon sign-up
MSI Authentication for egress to ADLS Gen1	Private Preview	Access granted upon sign-up

Built-in ML models for Anomaly Detection

Private Preview

NEW simpler function signatures for Anomaly Detection

```
AnomalyDetection_SpikeAndDip(
        <scalar_expression>,
        <confidence>,
        <historySize>,
        <mode>)
OVER
        ([PARTITION BY <partition key>]
        LIMIT DURATION(<unit>, <length>)
[WHEN
        boolean_expression])
```

```
AnomalyDetection_ChangePoint(
        <scalar_expression>,
        <confidence>,
        <historySize>)
OVER
        ([PARTITION BY <partition key>]
        LIMIT DURATION(<unit>, <length>)
[WHEN
        boolean_expression])
```

Output schema

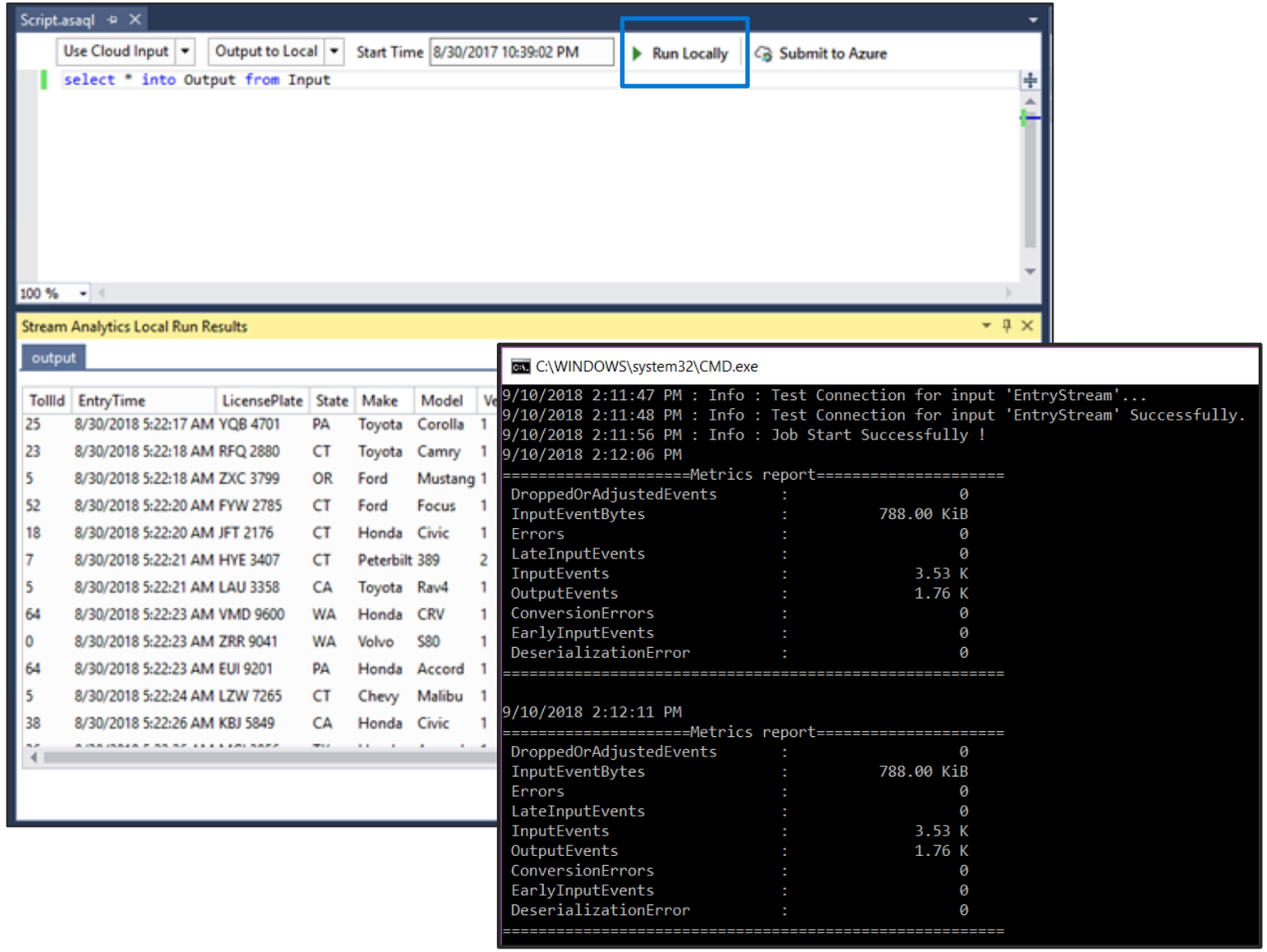
IsAnomaly: A bigint (0 or 1) indicating if the event was anomalous or not.

Score: A/D score (float) indicating how anomalous an event is.

Local testing with live data in Visual Studio

Public Preview

- Faster iterative testing
- Show results in real time
- View Job metrics
- Time policies support



Custom output partitioning to Blob Storage

Private Preview

Partition egress to Blob storage by

1) Any input field
2) Custom date and time formats

Gain more fine grain control over data written to Blob storage for dashboarding and reporting

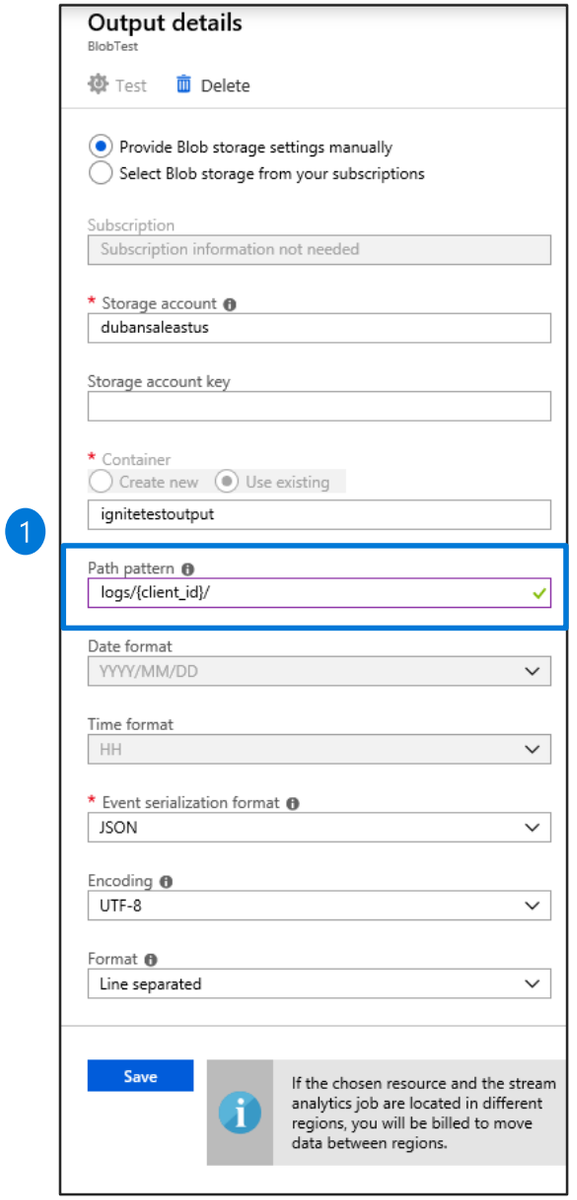
Better alignment with Hive conventions for blob output to be consumed by HDInsight and Azure Databricks.

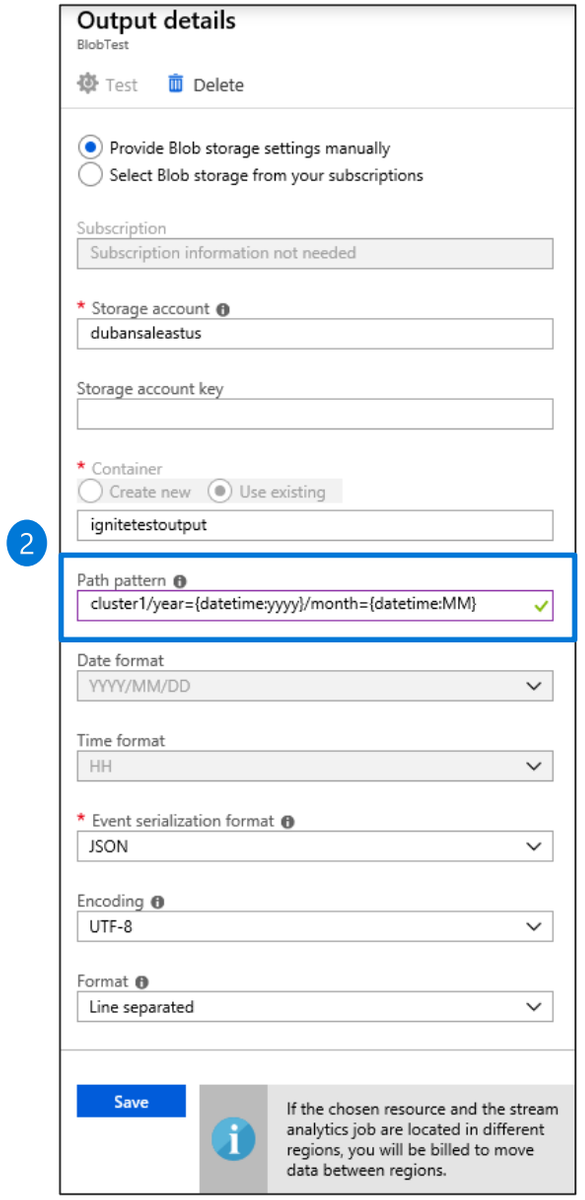
Path pattern:

```
year={datetime:yyyy}/month={datetime:MM}/day={datetime:dd}/hour={datetime:HH}
```

Azure blob storage folder nesting example:

```
year=2018
        month=07
                day=31
                        hour=05
                        hour=06
```





Hi-throughput output to SQL

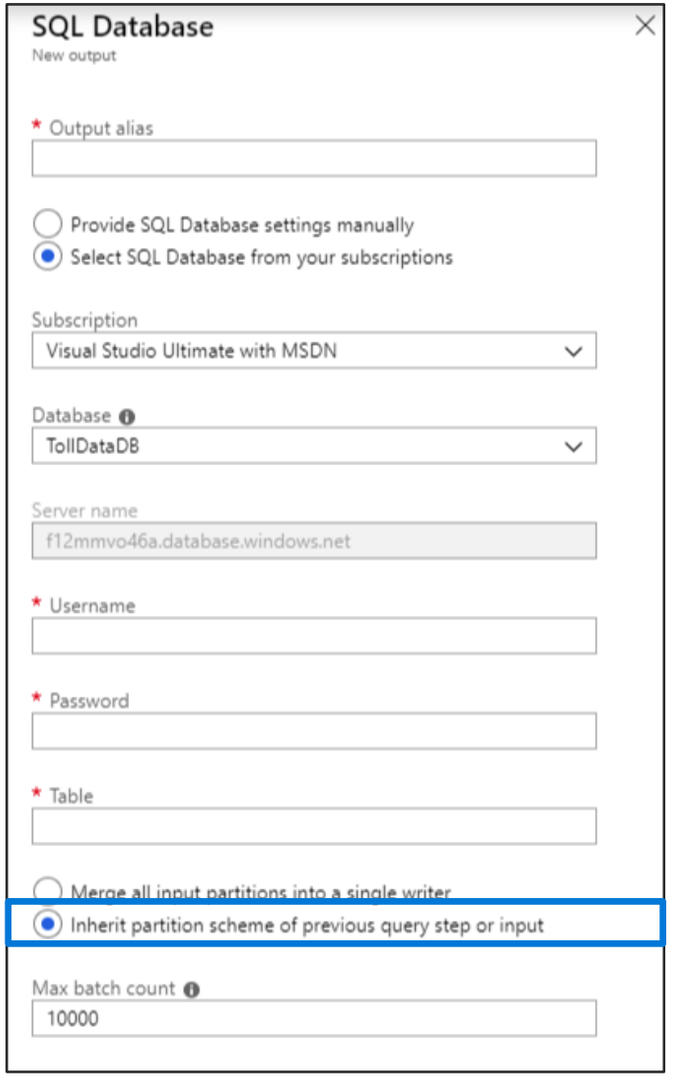
GA

To achieve fully parallel topologies, ASA will transition SQL ‘writes’ from Serial to Parallel operations for SQL DB and SQL Data Warehouse

4x-5x improvement in write throughput

Allows for batch size customization to achieve higher throughput

For e.g., this feature enabled our customer building a connected car scenario to scale up from 150K events/min to 500K events/min



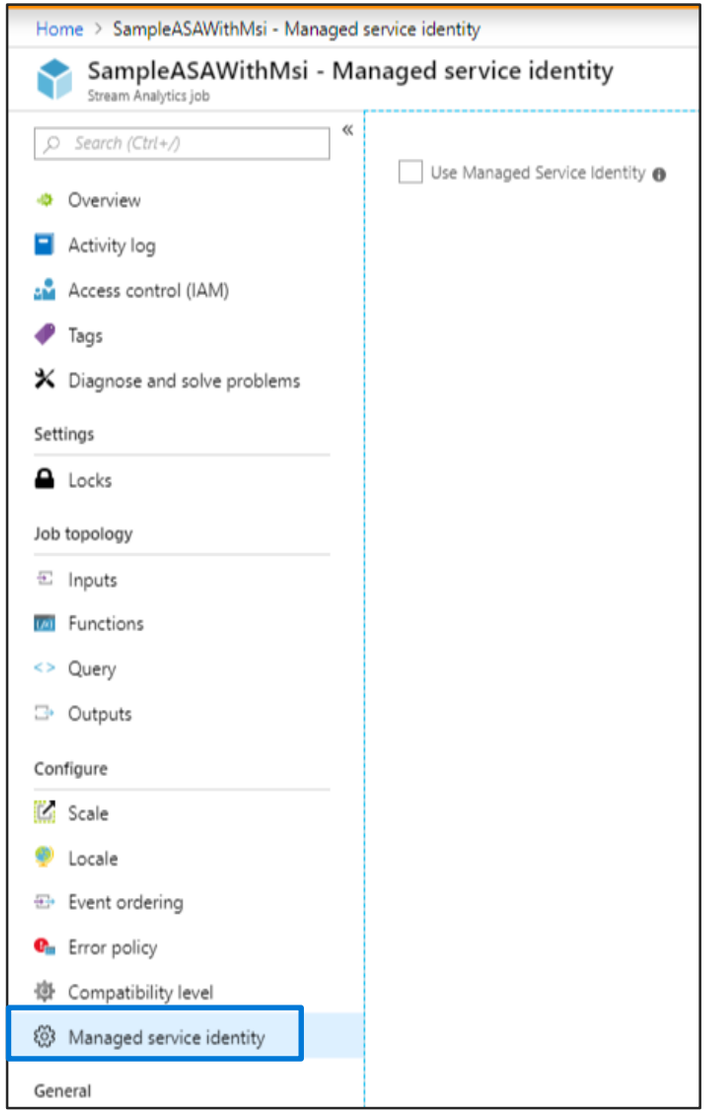
ADLS Service Principal Authentication

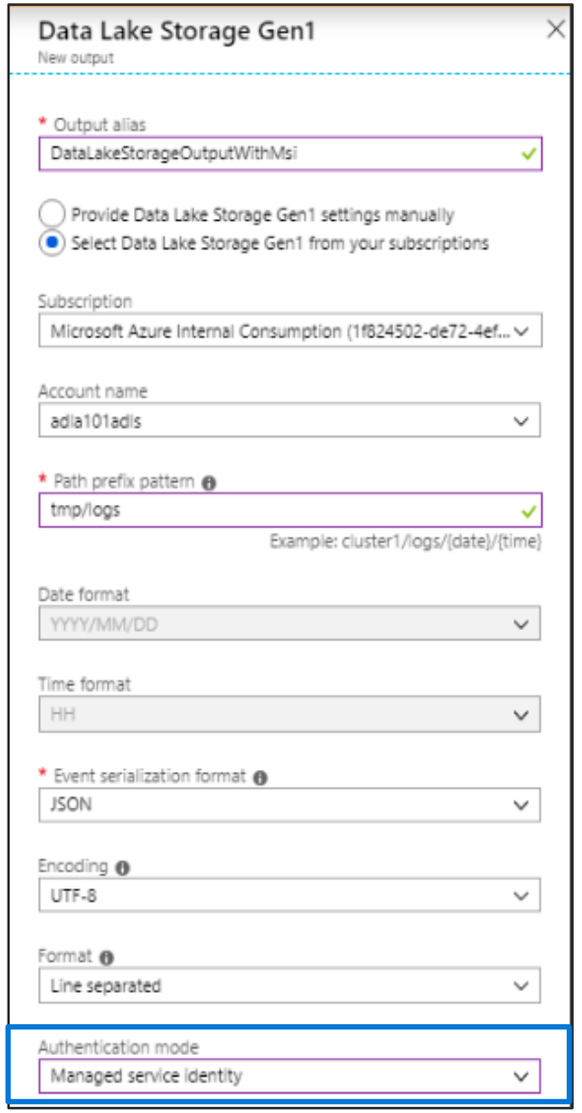
Private Preview

MSI based authentication will enable egress to Azure Data Lake Storage.

Key benefits over existing AAD (Azure Active Directory) based authentication:

- Job deployment automation (thru Power Shell etc.)
- Long running production jobs
- Consistency with other services





User specified custom re-partition count

Public Preview

Enables better performance tuning

Key Scenarios

- When upstream partition count can’t be changed
- Partitioned processing is needed to scale out to larger processing load
- Fixed number of output partitions

```
SELECT *
INTO
        [output]
FROM
        [input]
PARTITION BY
        DeviceID INTO 10
```

C# UDF support on Azure IoT Edge

Public Preview

Easily extend existing query language with C# User Defined Functions to enable new possibilities, including:

- Complex math calculations
- Machine learning on Azure Edge w/ ML.NET
- String/Date manipulations
- Custom data imputation logic
- Debugging experience in Visual Studio

```
1   using System;
2   using System.Collections.Generic;
3   using System.IO;
4   using System.Linq;
5   using System.Text;
6
7   namespace ASAEdgeUDFDemo
8   {
        0 references
9       public class Class1
10      {
11          // Add your public static function
            0 references
12          public static Int64 SquareFunction(Int64 a)
13          {
14              return a * a;
15          }
16      }
17  }
18
```

Run Locally Submit to Azure

```
Select machine.temperature,
udf.ASAEdgeUDFDemo_Class1_SquareFunction(try_cast(machine.tecommperature as bigint))
into Output
from Input
```

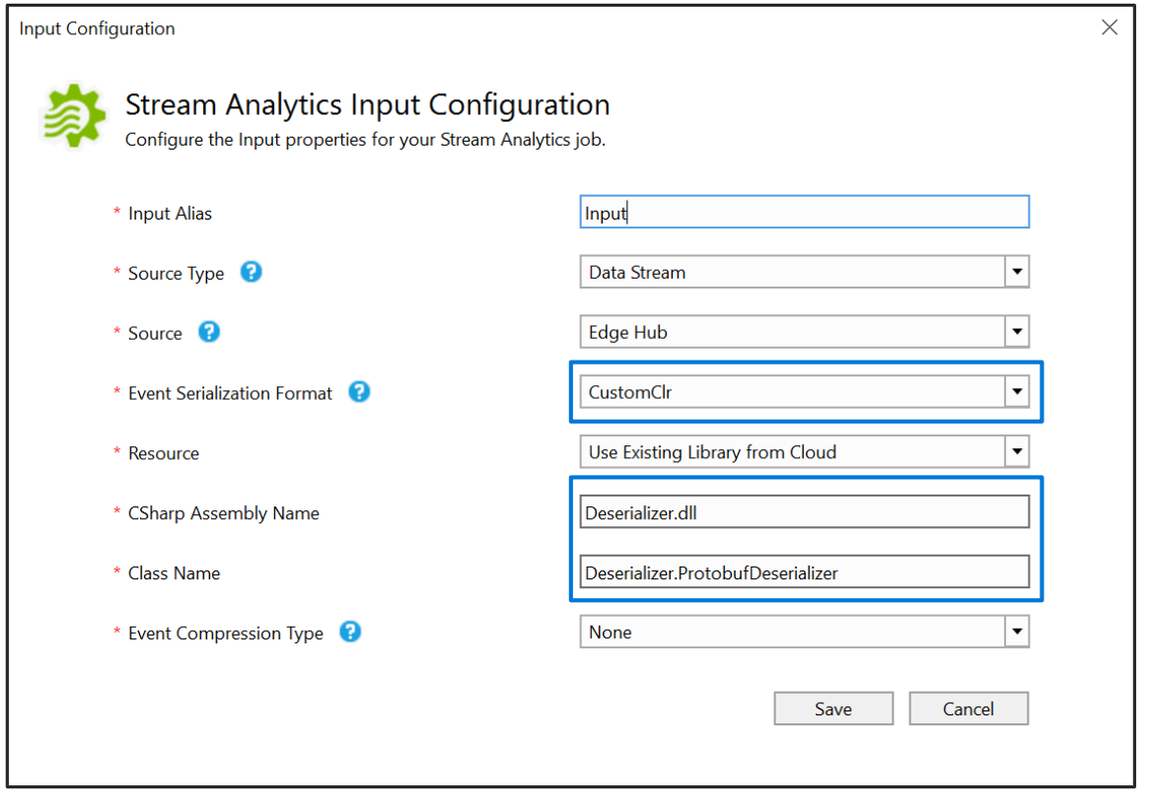
Custom de-serializers on Azure IoT Edge

Private Preview

Increases relevancy of Azure Stream Analytics across scores of new scenarios by extending support for data formats beyond JSON, CSV and AVRO

Use custom de-serializers to support a variety of data formats including:

- Protobuf
- Parquet
- XML
- Any binary format!!



Interacting with Stream Analytics

Modalities of Interaction & Programmability

Azure Portal

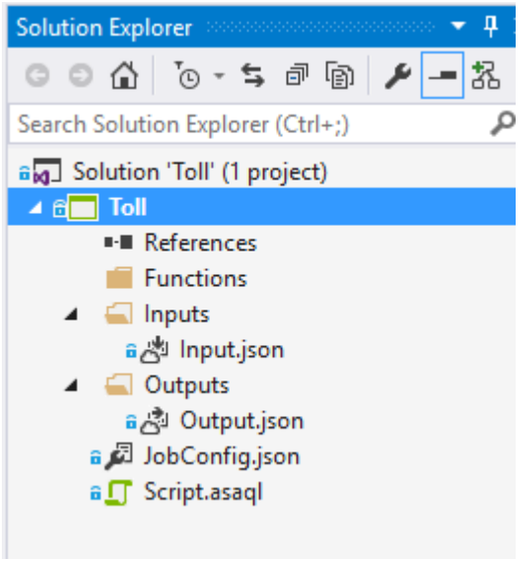
Visual Studio

PowerShell

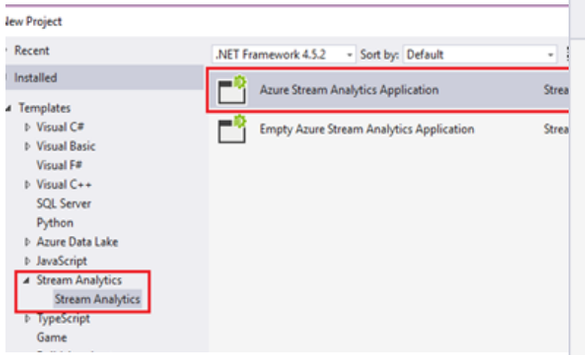
.NET SDK

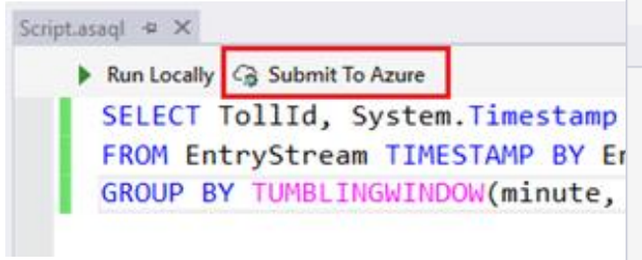
REST APIs

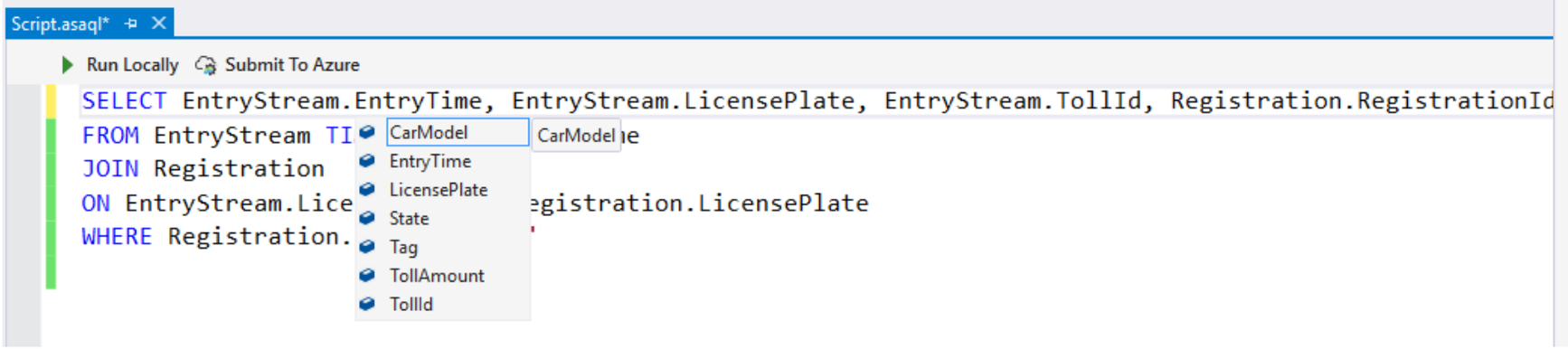
Visual Studio Support

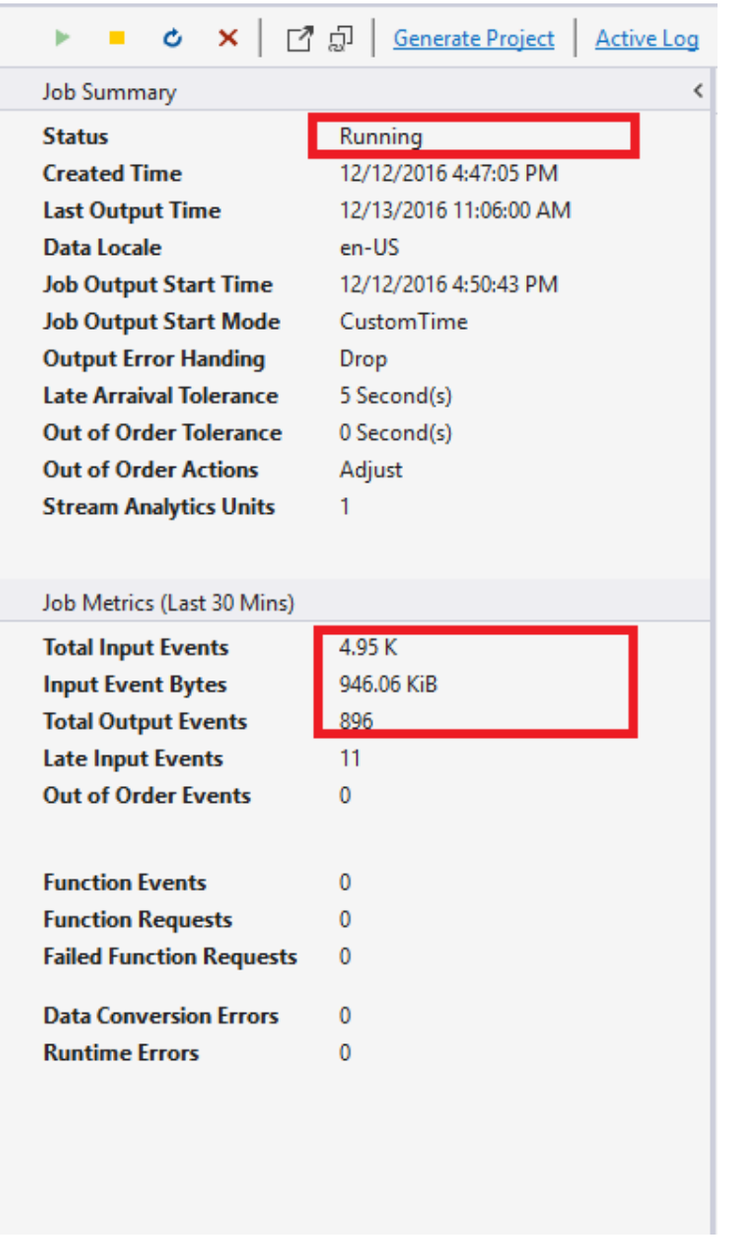


Version Control
Author with Intellisense
Test Locally
Deploy to Azure
Monitor









Q & A


Thank You!
ευχαριστώ
Salamat Po
متشكرم
شكراً
Grazie
благодаря
ありがとうございます
Kiitos
Teşekkürler
谢谢
ขอบคุณครับ
Obrigado
شكريه
Terima Kasih
Dziękuję
Hvala
Köszönöm
Tak
Dank u Wel
дякую
Tack
Mulţumesc
спасибо
Danke
Cám ơn
Gracias
多謝晒
Ďakujem
תודה
நன்றி
Děkuji
감사합니다