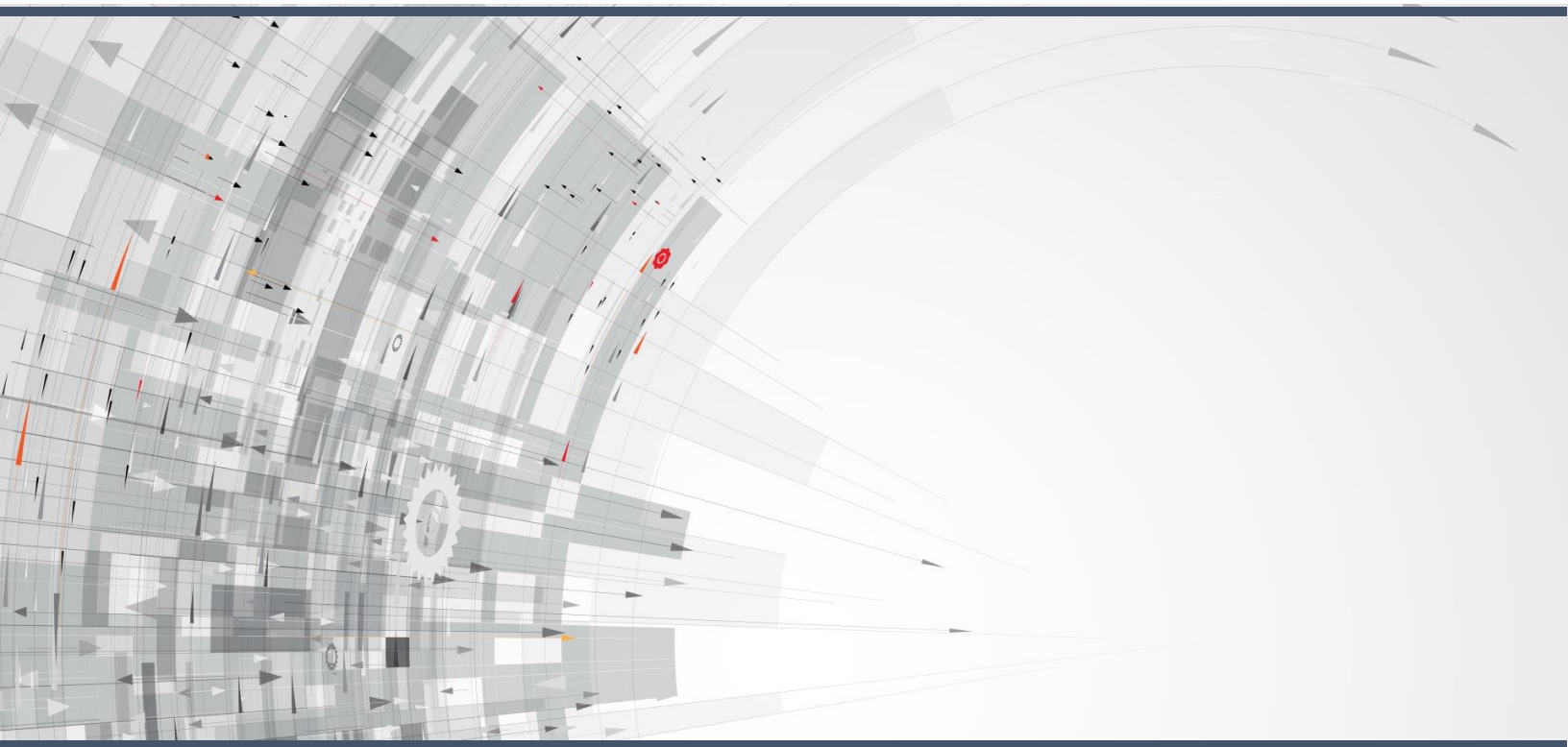
Wintellect
Know how.



Understanding
Azure Stream Analytics

Process high-volume, high-velocity data streams from IoT devices and other sources and extract information as easily as querying a database

October 2016

Today, roughly 20 billion devices are connected to the Internet. Depending on whose estimates you believe, by 2020 there will be 50 to 100 billion Internet-of-Things (IoT) sensors, mobile phones, and other devices flooding the Internet with data. The challenge for developers and data scientists alike is to harness that data and extract meaningful information from it. But while the science of querying databases is well understood, extracting information from fast-moving, constantly changing data streams presents challenges that require new and innovative approaches to collection and analysis.

Azure Stream Analytics is a cloud-based service for ingesting high-velocity data streaming from devices, sensors, applications, Web sites, and other data sources and analyzing that data in real time. It supports a SQL-like query language that works over dynamic data streams and makes analyzing constantly changing data no more difficult than querying traditional databases. With Azure Stream Analytics, you can set up jobs that analyze incoming data for anomalies or information of interest and record the results, present real-time notifications on dashboards, or fire off alerts to mobile devices. And all of it can be done at low cost and with minimal effort.

Scenarios for the application of real-time data analytics are legion and include fraud detection, analysis of trends in stock trading, resource-allocation optimization (think of a ride-sharing service that sends drivers to areas of increasing demand before that demand peaks), click-stream analysis on Web sites, preemptive maintenance on jet engines and wind turbines, and countless others. Having the ability to process data as it comes in rather than waiting until it has been aggregated offers a competitive advantage to businesses that are agile enough to make adjustments on the fly. And it enables us to understand the world around us by treating streaming data no differently than static data.

Stream Processing

Processing high-volume, highly dynamic data streams in real time presents challenges. Most software developers and data scientists know how to write a SQL query to extract information from a database, but databases are relatively static. How do you extract information from a live data stream that is constantly changing? Moreover, how do you query a data stream for temporal information? A SQL database tells you how many red cars are in your fleet. A stream processing system tells you how many red cars pass through a toll booth every 5 minutes, or how many, on average, pass through each day.

One solution to stream processing is Apache Storm, a free and open-source real-time computation system that does for stream processing what Apache Hadoop does for batch processing. Storm enjoys significant mindshare in the big-data community and is scalable (it has been clocked at more than a million tuples per second per node), fault-tolerant, and able to

integrate with existing queueing and database technologies. It can be enhanced with Storm SQL, which leverages Apache Calcite to support SQL queries on real-time data streams. But Storm is far from trivial to set up and maintain, especially if you're also responsible for the hardware clusters that host it.

Enter Azure Stream Analytics, which requires no setup and maintenance because it is as offered as Software-as-a-Service (SaaS). A few button clicks in the Azure Portal are sufficient to create a Stream Analytics job, configure it with inputs and outputs, and execute queries. It is fast and super-scalable; a single Stream Analytics job can handle millions of events per second. It integrates seamlessly with other Azure services such as Azure Storage, Azure SQL Database, and Microsoft Power BI. And SQL support is built in, enabling you to use a language you already know to analyze data in real time in order to trigger alerts, write to databases, update dashboards, and more.

> "We can have a new streaming job running in production in 10 minutes, which is several orders of magnitude faster than we were able to do it in the past." – *Ashley Noble, Honeywell Engineering Fellow*

Azure Stream Analytics

Figure 1 shows a typical real-time processing system built around Azure Stream Analytics. On the left are the sensors, devices, and other data sources that stream events to Azure Event Hubs, which can handle millions of events per second and deliver them to multiple applications. Data flows to Event Hubs directly via HTTPS and REST or the Advanced Message Queueing Protocol (AMQP) from data sources that are IP-capable, or indirectly via cloud gateways from data sources that are not. Because the Event Hub is connected to a Stream Analytics job as an input, all the events reaching the Event Hub flow through to Stream Analytics, where the data is queried and transformed. Output can be directed to a variety of endpoints to provide interactive views of the query results and, if desired, to capture it in durable data stores.

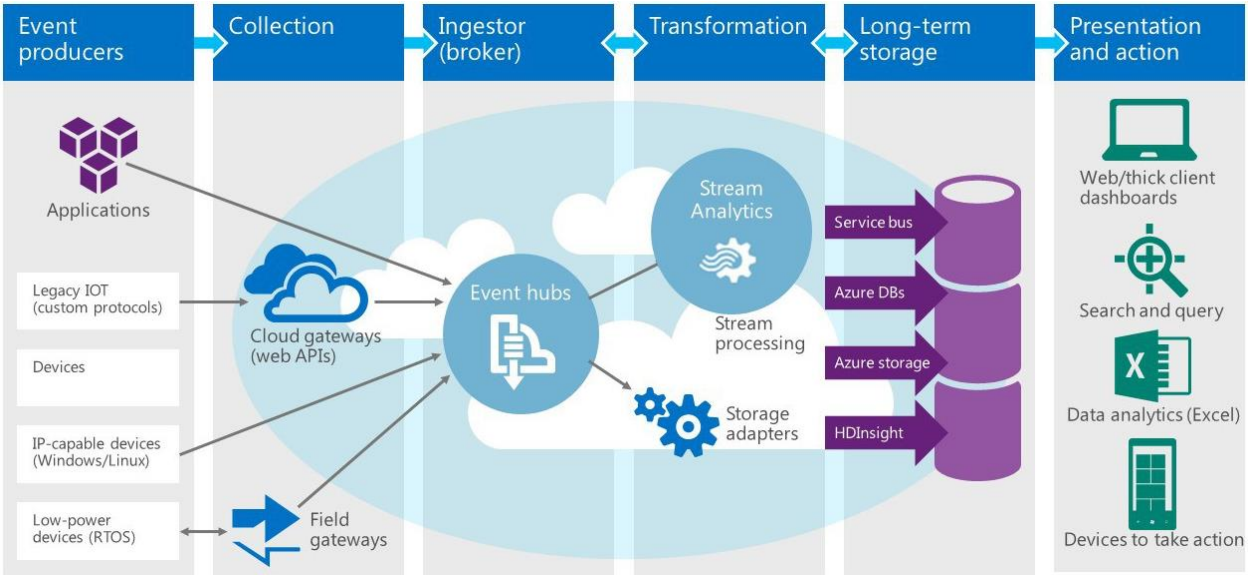


Figure 1: Real-time data processing with Azure Stream Analytics

Inputs to a Stream Analytics job are configured in the Azure Portal. Inputs can include Azure Event Hubs, Azure IoT Hubs (specialized Event Hubs with features added for IoT devices), and Azure Blob Storage. A single Stream Analytics job can have multiple inputs. Once inside Stream Analytics, data streams can be merged for the purpose of querying the combined data (similar to doing a JOIN in SQL). Stream Analytics also supports static reference data as input, which is useful for testing queries against known datasets as well as for joining to streaming data.

The outputs from a Stream Analytics job are also specified in the portal. Outputs can include any combination of the following:

- Azure SQL Database
- Azure Blob Storage
- Azure Table Storage
- Azure Event Hub
- Azure Service Bus Topic
- Azure Service Bus Queue
- Azure DocumentDB
- Microsoft Power BI
- Azure Data Lake Store

The wide range of output types that are supported enables a variety of scenarios. For example, a Stream Analytics job that monitors data streams from ATM machines could analyze the data for potentially fraudulent transactions. It could create a durable record of those transactions in

Azure Blob Storage, Azure Table Storage, Azure SQL Database, or Azure DocumentDB, and it could be paired with Microsoft Power BI to implement a real-time dashboard highlighting those transactions. Or it could direct its output to an Azure Event Hub, for which a custom dashboard application implemented in C# or another programming language could subscribe to events and send a push notification to a mobile device each time a suspicious transaction is identified.

The fact that a single Stream Analytics job can be configured with multiple inputs and outputs enables rich topologies comprising end-to-end stream-processing solutions. Given that a single Azure Event Hub can ingest millions of events per second and a Stream Analytics job can ingest data from multiple Event Hubs, is Azure Stream Analytics scalable enough to handle potentially massive data flows?

The capacity of a Stream Analytics job is measured in Streaming Units, or SUs. Each SU corresponds to roughly 1 MB/sec of throughput. By default, a Stream Analytics job is allocated 1 SU, but through the Azure Portal, this can be increased to up to 48 SUs. For more information about using Streaming Units to maximize throughput, see Scale Azure Stream Analytics jobs to increase stream data processing throughput.

The Stream Analytics Query Language

The heart of Azure Stream Analytics is the Stream Analytics Query Language (SAQL), which is derived from the ubiquitous SQL query language and features enhancements that are unique to stream processing. Queries can be formulated and tested in the Azure Portal (Figure 2) and then deployed against live data streams.

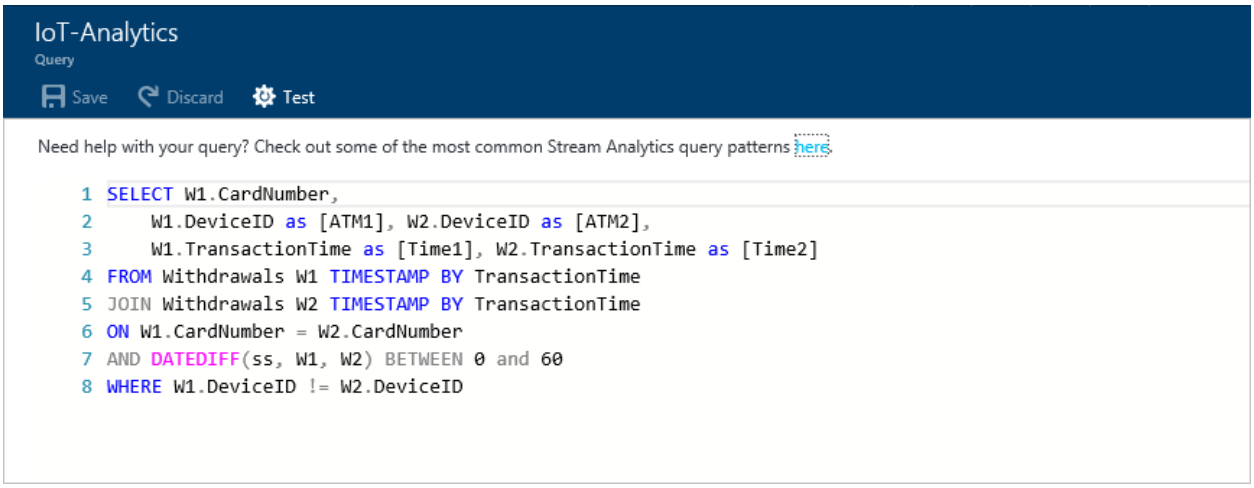


Figure 2: A SAQL query in the Azure Portal

> "The queries we need to run are quite complicated. We are able to do this much quicker with Stream Analytics, and with very low overhead." — *Arvind Shetty, Honeywell Technology Specialist*

For the sake of example, imagine that you manage a collection of toll booths, and that the toll booths transmit events to a Stream Analytics job via Azure Event Hubs each time a car enters or leaves. Picture two data streams – one for cars entering toll booths, and another for cars leaving – and two Event Hubs, each connected to Stream Analytics. Think of each event as a row in a database containing fields identifying the make and model of the car, the license-plate number, the time at which the car entered or left the toll booth (the "event time"), and more.

If the data stream representing cars entering a toll booth is named "EntryData" (a name that is assigned in the Azure Portal when the Event Hub is added as an input to the Stream Analytics job), then the following query lists all cars with Connecticut license plates that enter a toll booth:

```
SELECT EntryTime, TollId, LicensePlate
FROM EntryData
WHERE State = 'CT'
```

The next query goes further: it JOINs the two data streams ("EntryData" and "ExitData") and computes the difference between entry time and exit time for every car using the DATEDIFF function. The output lists each car that enters and exits a toll booth, along with the toll booth ID, the entry time, and the elapsed time:

```
SELECT EN.TollId, EN.EntryTime, EN.LicensePlate,
   DATEDIFF(minute, EN.EntryTime, EX.ExitTime) AS Minutes
FROM EntryData EN TIMESTAMP BY EntryTime
JOIN ExitData EX TIMESTAMP BY ExitTime
   ON EN.TollId = EX.TollId
   AND EN.LicensePlate = EX.LicensePlate
   AND DATEDIFF(minute, EN, EX) BETWEEN 0 AND 60
```

In addition to showing how JOIN may be used to merge two streams (a powerful feature that is analogous to joining two tables in a database), this example demonstrates one of the important extensions that SAQL adds to the SQL query language: the **TIMESTAMP BY** keyword. By default, functions such as **DATEDIFF** use the event arrival time – the time at which the event arrived at

the Event Hub or other input source – to perform temporal calculations. **TIMESTAMP BY** designates a field in the input stream as the event time (the time at which the event actually occurred rather than the time at which it was recorded) for accurate computations that are independent of lag time, caching performed upstream of the Event Hub, and other factors. A related extension keyword named **System.Timestamp** allows the event time to be referenced in queries.

The Stream Analytics Query Language is a subset of Microsoft's T-SQL. It supports many of the same data types, includes many of the same functions, and supports many of the same query-language elements, including **SELECT**, **JOIN**, **UNION**, **WITH**, **GROUP BY**, and many others. For a comprehensive overview of the language and the constructs that it supports, see the Stream Analytics Query Language Reference.

Windowing

One of the key features of the Stream Analytics Query Language is its ability to group results using windows of time whose length you specify. Syntactically, you exercise windowing by using SQL's **GROUP BY** clause with the extension keywords **TumblingWindow**, **HoppingWindow**, and **SlidingWindow**. **TumblingWindow** allows you to ask questions such as "How many red cars enter my toll booths every 5 minutes?" **HoppingWindow** lets you ask the same question but generate output at intervals that are independent of the window size – for example, "Tell me once a minute how many red cars enter my toll booths every 5 minutes." **SlidingWindow** lets you ask "During which 5-minute time periods do 10 or more red cars go through my toll booths?" and is ideal for dealing with relatively sparse result sets. These concepts are illustrated in Figure 3.

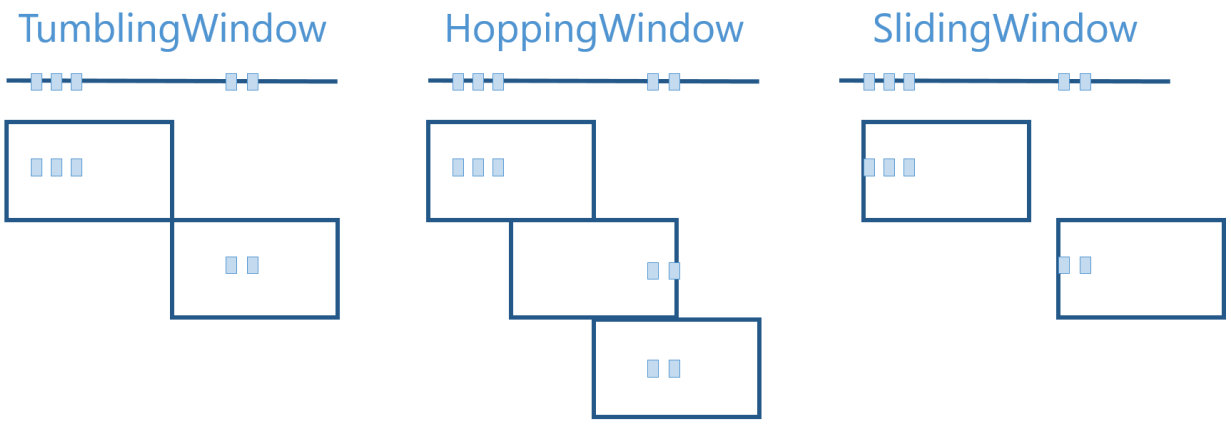


Figure 3: Grouping results using windows

The following query counts the number of cars with NY license plates entering toll booths every 5 minutes and produces one output (one count) every 5 minutes:

```
SELECT DateAdd(minute, -5, System.TimeStamp) AS [Start Time],
   System.TimeStamp AS [End Time], COUNT(*)
FROM EntryData TIMESTAMP BY EntryTime
WHERE State = 'NY'
GROUP BY TumblingWindow(minute, 5)
```

The next query computes the average wait time (the difference between the time the car enters and exits a toll both) for all cars for the last 5 minutes. The same query performed with **TumblingWindow** would produce an output every 5 minutes. Because it uses **HoppingWindow** with a final parameter of 1, this query produces an output every 1 minute instead:

```
SELECT DateAdd(minute, -5, System.TimeStamp) AS [Start Time],
   System.TimeStamp AS [End Time],
   AVG(DATEDIFF(minute, EN.EntryTime, EX.ExitTime)) AS [Average Wait Time]
FROM EntryData EN TIMESTAMP BY EntryTime
JOIN ExitData EX TIMESTAMP BY ExitTime
   ON EN.TollId = EX.TollId
   AND EN.LicensePlate = EX.LicensePlate
   AND DATEDIFF(minute, EN, EX) BETWEEN 0 AND 60
GROUP BY HoppingWindow(minute, 5, 1)
```

Finally, this query lists each 5-minute time period in which at least one car with NJ plates enters a toll booth:

```
SELECT DateAdd(minute, -5, System.TimeStamp) AS [Start Time],
   System.TimeStamp AS [End Time], TollId, COUNT(*)
FROM EntryData TIMESTAMP BY EntryTime
WHERE State = 'NJ'
GROUP BY TollId, SlidingWindow(minute, 5)
HAVING COUNT(*) > 0
```

These are but a few examples of how windowing can be used to extract temporal information from live data streams. For more information, see Windowing (Azure Stream Analytics) on the Azure Web site. A related document, Query examples for common Stream Analytics query patterns, presents common query patterns and provides helpful guidance in formulating queries with the Stream Analytics Query Language.

Azure Stream Analytics in Action

Honeywell Building Systems put Azure Stream Analytics to work processing real-time information from the equipment that regulates air quality and temperature in the facilities that it serves, enabling the company to monitor environmental conditions, perform preemptive maintenance, and save its customers time, energy, and money.

In 2014, Honeywell began connecting its vast network of devices to an IoT platform so they could autonomously send data to remote monitors in the cloud. "Once we started lifting the data off-site, we tried using homegrown techniques, which let us analyze it only every 12 or 24 hours," says Honeywell Engineering Fellow Ashley Noble. "This did not meet some of our customers' needs." Honeywell customers needed to know how buildings were functioning—or malfunctioning—in real time.

Honeywell turned to the Microsoft Azure cloud platform. By participating in the Microsoft Rapid Development Program, a three-day technology-orientation preview, Honeywell engineers were able to quickly begin configuring a data-processing environment capable of ingesting thousands of events as they happened. The system compared those streams to models configured to reflect optimal operating conditions so they could detect errant behavior that would signify equipment failure, substandard performance, or disrepair.

To make this work, the engineers programmed each of the sensors in the field to send a stream of messages every minute describing the changes occurring in the equipment, building, and configuration state to the IoT platform. Azure Table Storage stores the telemetry data. Robust messaging and queuing infrastructure from Azure Service Bus Topics and Azure Event Hubs provide fault tolerance. Stream Analytics then identifies certain building-specific equipment behavior as fault-state behavior and tags it as an anomaly. When a specific error or condition appears in the incoming streams of data, Stream Analytics triggers the notification engine to send an alert to the customer via a customized dashboard. The Azure platform then stores alerts and faults so customers can perform further analysis to discover and predict trends and assist with related security or performance risks.

Because Honeywell Building Solutions could configure its data processing solution in the cloud, it had no hardware to deploy and could begin generating results in Stream Analytics almost immediately. "We had results within weeks, which is extraordinarily fast," says Noble. "And the cost of standing up a new processing solution is so low, we can easily try one out across our building data and see if it provides benefit to our customers."

For more on how Honeywell Building Systems used Azure Stream Analytics to better serve its customers and about other companies that are using it, see Innovative Azure Stream Analytics customer use cases.

Summary

Azure Stream Analytics simplifies the task of extracting information from fast-moving data streams and processing that information in real time. As part of the Azure data-analytics suite, Azure Stream Analytics integrates seamlessly with other Azure services and provides a ready-to-use solution for developers, researchers, and data scientists whose jobs require them to analyze data emanating from IoT devices and other data sources in real time.

Need Help with Azure Stream Analytics?