Table of Contents

- Overview
  - What is Stream Analytics?
  - Compare Storm & Stream Analytics
- Get started
  - Process IoT streaming data
  - Real-time fraud detection
- How to
  - Build streaming solutions
    - Internet of Things
    - Social media
    - Sensor data with Power BI
    - Real-time scoring with Machine Learning
    - Real-time event processing
  - Develop
    - Common query patterns
    - Tools for Visual Studio
    - Create inputs
    - Create outputs
    - Use reference data
    - Output to Azure Functions
    - Output to Data Lake Store
    - Output to DocumentDB
    - Analyze data with Power BI
    - JavaScript UDF integration
    - REST API & Machine Learning integration
    - Use the Management .NET SDK
    - Window functions
  - Manage
    - Monitor jobs

- Diagnostic logs
- Monitor jobs programmatically
- Monitor jobs with PowerShell
- Scale to increase throughput
- Scale for Machine Learning functions
- Rotate input & output credentials
- Automate
  - One-click PowerShell
- Reference
- PowerShell
- .NET
- Query language
- REST
- Resources
- Release notes
- Learning Path
- Pricing
- Blog
- Forum
- Stack Overflow
- Feedback forum
- Service updates
- Videos

What is Stream Analytics?

1/25/2017 • 4 min to read • Edit on GitHub

Azure Stream Analytics is a fully managed, cost effective real-time event processing engine that helps to unlock deep insights from data. Stream Analytics makes it easy to set up real-time analytic computations on data streaming from devices, sensors, web sites, social media, applications, infrastructure systems, and more.

With a few clicks in the Azure portal, you can author a Stream Analytics job specifying the input source of the streaming data, the output sink for the results of your job, and a data transformation expressed in a SQL-like language. You can monitor and adjust the scale/speed of your job in the Azure portal to scale from a few kilobytes to a gigabyte or more of events processed per second.

Stream Analytics leverages years of Microsoft Research work in developing highly tuned streaming engines for time-sensitive processing, as well as language integrations for intuitive specifications of such.

What can I use Stream Analytics for?

Vast amounts of data are flowing at high velocity over the wire today. Organizations that can process and act on this streaming data in real time can dramatically improve efficiencies and differentiate themselves in the market. Scenarios of real-time streaming analytics can be found across all industries: personalized, real-time stock-trading analysis and alerts offered by financial services companies; real-time fraud detection; data and identity protection services; reliable ingestion and analysis of data generated by sensors and actuators embedded in physical objects (Internet of Things, or IoT); web clickstream analytics; and customer relationship management (CRM) applications issuing alerts when customer experience within a time frame is degraded. Businesses are looking for the most flexible, reliable and cost-effective way to do such real-time event-stream data analysis to succeed in the highly competitive modern business world.

Key capabilities and benefits

- **Ease of use:** Stream Analytics supports a simple, declarative query model for describing transformations. In order to optimize for ease of use, Stream Analytics uses a T-SQL variant, and removes the need for customers to deal with the technical complexities of stream processing systems. Using the Stream Analytics query language in the in-browser query editor, you get intelli-sense auto-complete to help you can quickly and easily implement time series queries, including temporal-based joins, windowed aggregates, temporal filters, and other common operations such as joins, aggregates, projections, and filters. In addition, in-browser query testing against a sample data file enables quick, iterative development.
- **Scalability:** Stream Analytics is capable of handling high event throughput of up to 1GB/second. Integration with Azure Event Hubs and Azure IoT Hubs allow the solution to ingest millions of events per second coming from connected devices, clickstreams, and log files, to name a few. In order to achieve this, Stream Analytics leverages the partitioning capability of Event Hubs, which can yield 1MB/s per partition. Users are able to partition the computation into a number of logical steps within the query definition, each with the ability to be further partitioned to increase scalability.
- **Reliability, repeatability and quick recovery:** A managed service in the cloud, Stream Analytics helps prevent data loss and provides business continuity in the event of failures through built-in recovery capabilities. With the ability to internally maintain state, the service provides repeatable results ensuring it is possible to archive events and reapply processing in the future, always getting the same results. This enables customers to go back in time and investigate computations when doing root-cause analysis, what-if analysis, etc.
- **Low cost:** As a cloud service, Stream Analytics is optimized to provide users a very low cost to get going and

maintain real-time analytics solutions. The service is built for you to pay as you go based on Streaming Unit usage and the amount of data processed by the system. Usage is derived based on the volume of events processed and the amount of compute power provisioned within the cluster to handle the respective Stream Analytics jobs.

- **Reference data:** Stream Analytics provides users the ability to specify and use reference data. This could be historical data or simply non-streaming data that changes less frequently over time. The system simplifies the use of reference data to be treated like any other incoming event stream to join with other event streams ingested in real time to perform transformations.
- **User Defined Functions:** Stream Analytics has integration with Azure Machine Learning to define function calls in the Machine Learning service as part of a Stream Analytics query. This expands the capabilities of Stream Analytics to leverage existing Azure Machine Learning solutions. For further information on this, please review the Machine Learning integration tutorial.
- **Connectivity:** Stream Analytics connects directly to Azure Event Hubs and Azure IoT Hubs for stream ingestion, and the Azure Blob service to ingest historical data. Results can be written from Stream Analytics to Azure Storage Blobs or Tables, Azure SQL DB, Azure Data Lake Stores, DocumentDB, Event Hubs, Azure Service Bus Topics or Queues, and Power BI, where it can then be visualized, further processed by workflows, used in batch analytics via Azure HDInsight or processed again as a series of events. When using Event Hubs it is possible to compose multiple Stream Analytics together with other data sources and processing engines without losing the streaming nature of the computations.

Get help

For further assistance, try our Azure Stream Analytics forum

Next steps

You've been introduced to Stream Analytics, a managed service for streaming analytics on data from the Internet of Things. To learn more about this service, see:

- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Help choosing a streaming analytics platform: Apache Storm comparison to Azure Stream Analytics

1/25/2017 • 4 min to read • Edit on GitHub

Get guidance choosing a cloud analytics platform by using this Apache Storm comparison to Azure Stream Analytics. Understand the value propositions of Stream Analytics versus Apache Storm as a managed service on Azure HDInsight, so you can choose the right solution for your business use cases.

Both analytics platforms provide benefits of a PaaS solution, there are a few major distinguishing capabilities that differentiate them. Capabilities as well as the limitations of these services are listed below to help you land on the solution you need to achieve your goals.

Storm comparison to Stream Analytics: General features

	Azure Stream Analytics	**Apache Storm on HDInsight**
Open Source	No, Azure Stream Analytics is a Microsoft proprietary offering.	Yes, Apache Storm is an Apache licensed technology.
Microsoft Supported	Yes	Yes
Hardware requirements	There are no hardware requirements. Azure Stream Analytics is an Azure Service.	There are no hardware requirements. Apache Storm is an Azure Service.
Top Level Unit	With Azure Stream Analytics customers deploy and monitor streaming jobs.	With Apache Storm on HDInsight customers deploy and monitor a whole cluster, which can host multiple Storm jobs as well as other workloads (incl. batch).
Price	Stream Analytics is priced by volume of data processed and the number of streaming units (per hour the job is running) required. Further pricing information can be found here.	For Apache Storm on HDInsight, the unit of purchase is cluster-based, and is charged based on the time the cluster is running, independent of jobs deployed. Further pricing information can be found here.

Authoring on each analytics platform

	Azure Stream Analytics	**Apache Storm on HDInsight**
Capabilities: SQL DSL	Yes, an easy to use SQL language support is available.	No, users must write code in Java C# or use Trident APIs.

Capabilities: Temporal operators	Windowed aggregates, and temporal joins are supported out of the box.	Temporal operators must to be implemented by the user.
Development Experience	Interactive authoring and debugging experience through Azure Portal on sample data.	Development, debugging and monitoring experience is provided through the Visual Studio experience for .NET users, while for Java and other languages developers may use the IDE of their choice.
Debugging support	Stream Analytics offers basic job status and Operations logs as a way of debugging, but currently does not offer flexibility in what/how much is included in the logs ie: verbose mode.	Detailed logs are available for debugging purposes. There are two ways to surface logs to user, via visual Studio or user can RDP into the cluster to access logs.
Support for UDF (User Defined Functions)	Currently there is no support for UDFs.	UDFs can be written in C#, Java or the language of your choice.
Extensible - custom code	There is no support for extensible code in Stream Analytics.	Yes, there is availability to write custom code in C#, Java or other supported languages on Storm.

Data sources and outputs

	Azure Stream Analytics	**Apache Storm on HDInsight**
Input Data Sources	The supported input sources are Azure Event Hubs and Azure Blobs.	There are connectors available for Event Hubs, Service Bus, Kafka, etc. Unsupported connectors may be implemented via custom code.
Input Data formats	Supported input formats are Avro, JSON, CSV.	Any format may be implemented via custom code.
Outputs	A streaming job may have multiple outputs. Supported Outputs: Azure Event Hubs, Azure Blob Storage, Azure Tables, Azure SQL DB, and PowerBI.	Support for many outputs in a topology, each output may have custom logic for downstream processing. Out of the box Storm includes connectors for PowerBI, Azure Event Hubs, Azure Blob Store, Azure DocumentDB, SQL and HBase. Unsupported connectors may be implemented via custom code.
Data Encoding formats	Stream Analytics requires UTF-8 data format to be utilized.	Any data encoding format may be implemented via custom code.

Management and operations

	Azure Stream Analytics	Apache Storm on HDInsight
Job Deployment model - **Azure Portal** - **Visual Studio** - **PowerShell**	Deployment is implemented via Azure Portal, PowerShell and REST APIs.	Depolyment is implemented via Azure Portal, PowerShell, Visual Studio and REST APIs.
Monitoring (stats, counters, etc.)	Monitoring is implemented via Azure Portal and REST APIs. The user may also configure Azure alerts.	Monitoring is implemented via Storm UI and REST APIs.
Scalability	Number of Streaming Units for each job. Each Streaming Unit processes up to 1MB/s. Max of 50 units by default. Call to increase limit.	Number of nodes in the HDI Storm cluster. No limit on number of nodes (Top limit defined by your Azure quota). Call to increase limit.
Data processing limits	Users can scale up or down number of Streaming Units to increase data processing or optimize costs. Scale up to 1 GB/s	User can scale up or down cluster size to meet needs.
Stop/Resume	Stop and resume from last place stopped.	Stop and resume from last place stopped based on the watermark.
Service and framework update	Automatic patching with no downtime.	Automatic patching with no downtime.
Business continuity through a Highly Available Service with guaranteed SLA's	SLA of 99.9% uptime Auto-recovery from failures Recovery of stateful temporal operators is built-in.	SLA of 99.9% uptime of the Storm cluster. Apache Storm is a fault tolerant streaming platform however it is the customers' responsibility to ensure their streaming jobs run uninterrupted.

Advanced Features

	Azure Stream Analytics	Apache Storm on HDInsight
Late arrival and out of order event handling	Built-in configurable policies to reorder, drop events or adjust event time.	User must implement logic to handle this scenario.

Reference data	Reference data available from Azure Blobs with max size of 100 MB of in-memory lookup cache. Refreshing of reference data is managed by the service.	No limits on data size. Connectors available for HBase, DocumentDB, SQL Server and Azure. Unsupported connectors may be implemented via custom code. Refreshing of reference data must be handled by custom code.
Integration with Machine Learning	By configuring published Azure Machine Learning models as functions during ASA job creation (private preview).	Available through Storm Bolts.

Get started with Azure Stream Analytics to process data from IoT devices

1/23/2017 • 5 min to read • Edit on GitHub

In this tutorial, you will learn how to create stream-processing logic to gather data from Internet of Things (IoT) devices. We will use a real-world, Internet of Things (IoT) use case to demonstrate how to build your solution quickly and economically.

Prerequisites

- Azure subscription
- Sample query and data files downloadable from GitHub

Scenario

Contoso, which is a company in the industrial automation space, has completely automated its manufacturing process. The machinery in this plant has sensors that are capable of emitting streams of data in real time. In this scenario, a production floor manager wants to have real-time insights from the sensor data to look for patterns and take actions on them. We will use the Stream Analytics Query Language (SAQL) over the sensor data to find interesting patterns from the incoming stream of data.

Here data is being generated from a Texas Instruments sensor tag device.

The payload of the data is in JSON format and looks like the following:

```
{
    "time": "2016-01-26T20:47:53.0000000",
    "dspl": "sensorE",
    "temp": 123,
    "hmdt": 34
}
```

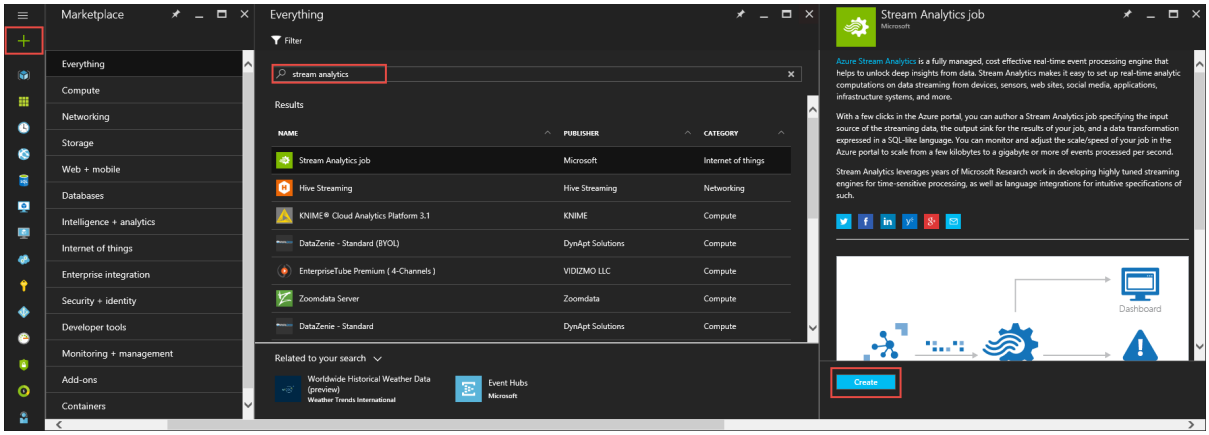
In a real-world scenario, you could have hundreds of these sensors generating events as a stream. Ideally, a gateway device would run code to push these events to Azure Event Hubs or Azure IoT Hubs. Your Stream Analytics

job would ingest these events from Event Hubs and run real-time analytics queries against the streams. Then, you could send the results to one of the supported outputs.

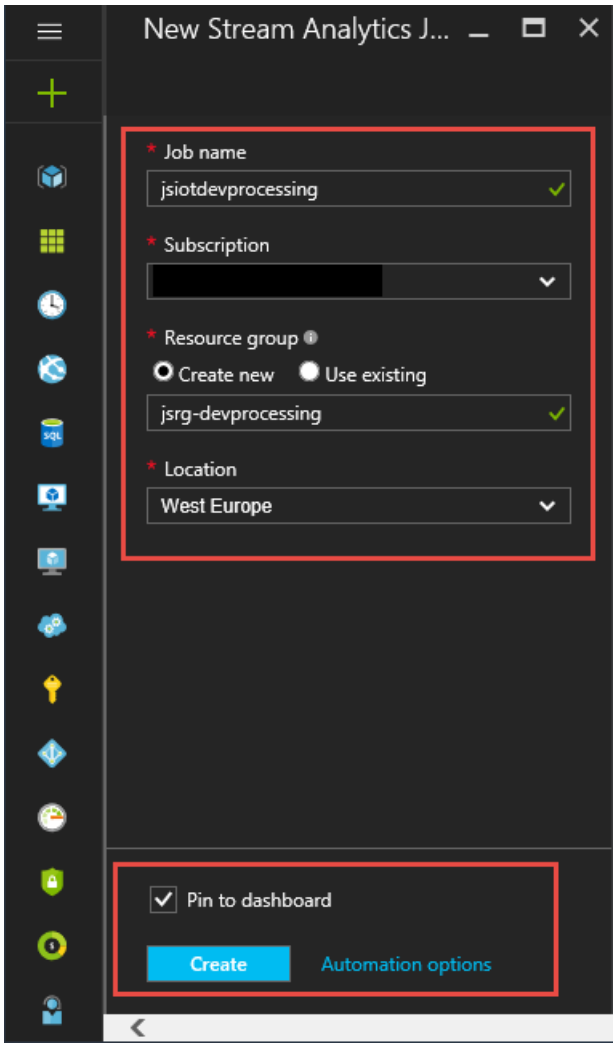
For ease of use, this getting started guide provides a sample data file, which was captured from real sensor tag devices. You can run queries on the sample data and see results. In subsequent tutorials, you will learn how to connect your job to inputs and outputs and deploy them to the Azure service.

Create a Stream Analytics job

1. In the Azure portal, click the plus sign and then type **STREAM ANALYTICS** in the text window to the right. Then select **Stream Analytics job** in the results list.

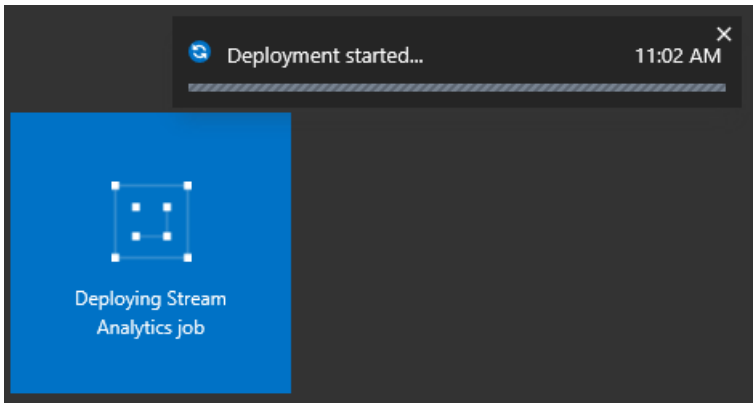


2. Enter a unique job name and verify the subscription is the correct one for your job. Then either create a new resource group or select an existing one on your subscription.
3. Then select a location for your job. For speed of processing and reduction of cost in data transfer selecting the same location as the resource group and intended storage account is recommended.

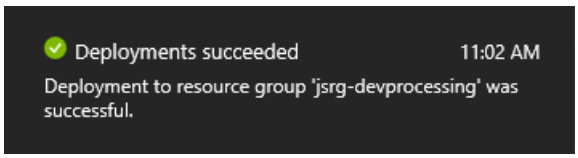


> **NOTE**
> You should create this storage account only once per region. This storage will be shared across all Stream Analytics jobs that are created in that region.

4. Check the box to place your job on your dashboard and then click **CREATE**.

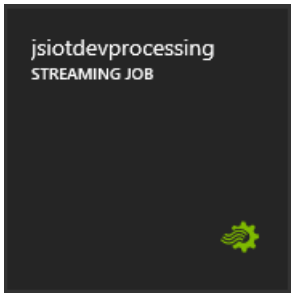


5. You should see a 'Deployment started...' displayed in the top right of your browser window. Soon it will change to a completed window as shown below.

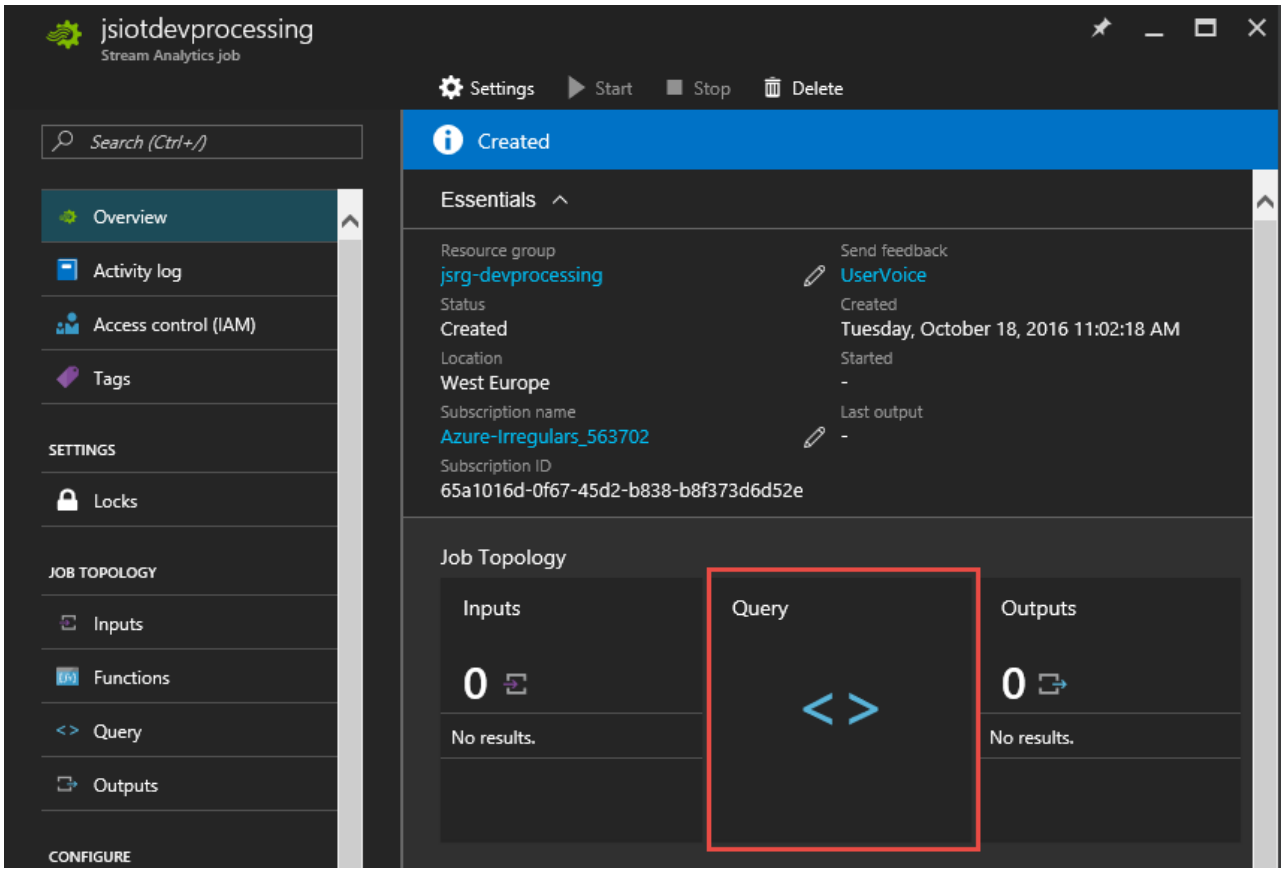


Create an Azure Stream Analytics query

After your job is created it's time to open it and build a query. You can easily access your job by clicking the tile for it.



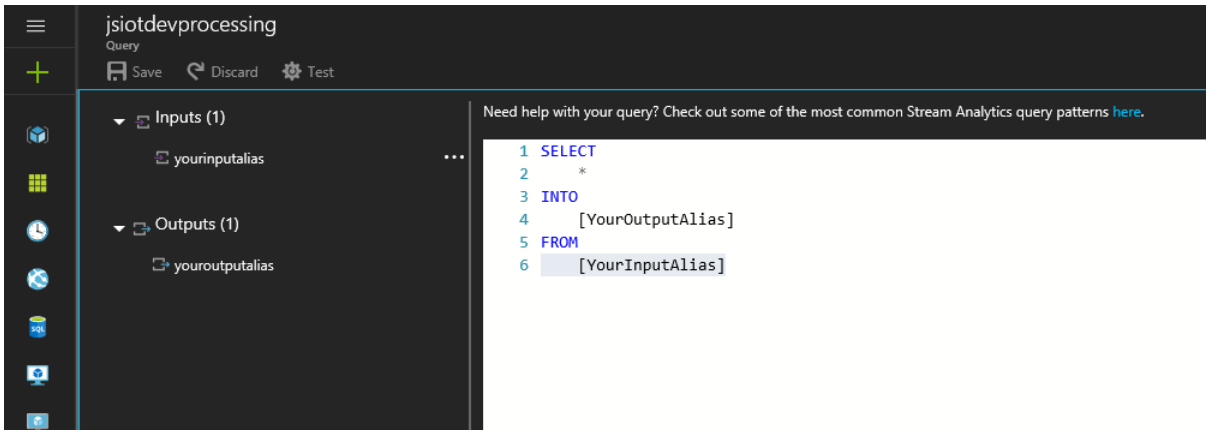
In the **Job Topology** pane click the **QUERY** box to go to the Query Editor. The **QUERY** editor allows you to enter a T-SQL query that performs the transformation over the incoming event data.



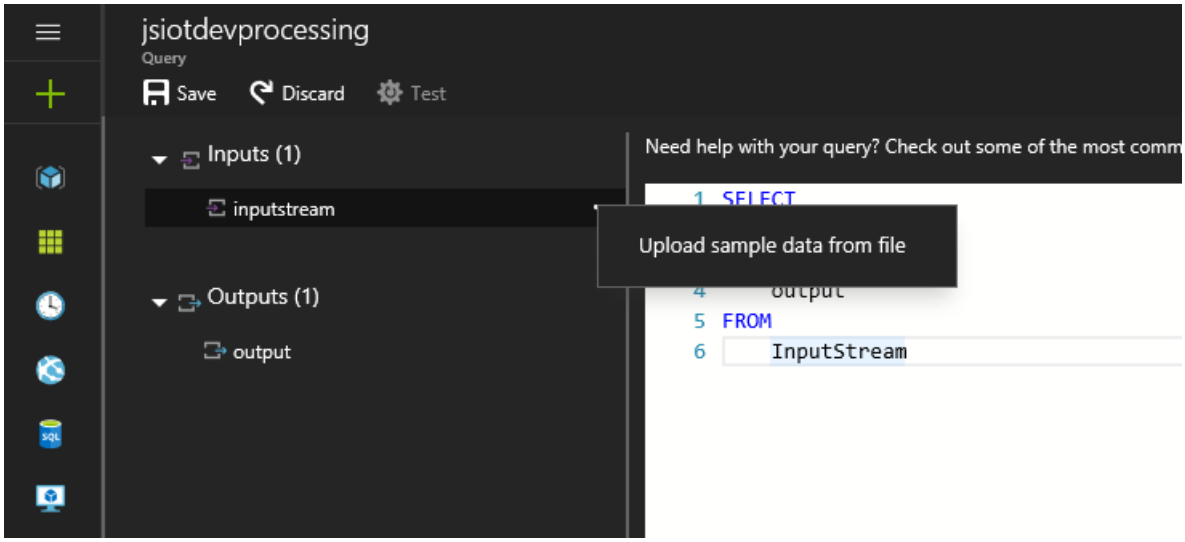
Query: Archive your raw data

The simplest form of query is a pass-through query that archives all input data to its designated output. Download the sample data file from GitHub to a location on your computer.

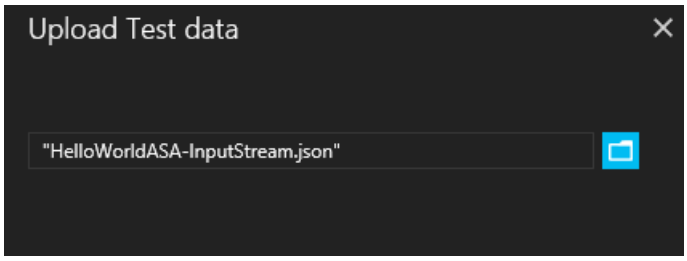
1. Paste the query from the PassThrough.txt file.



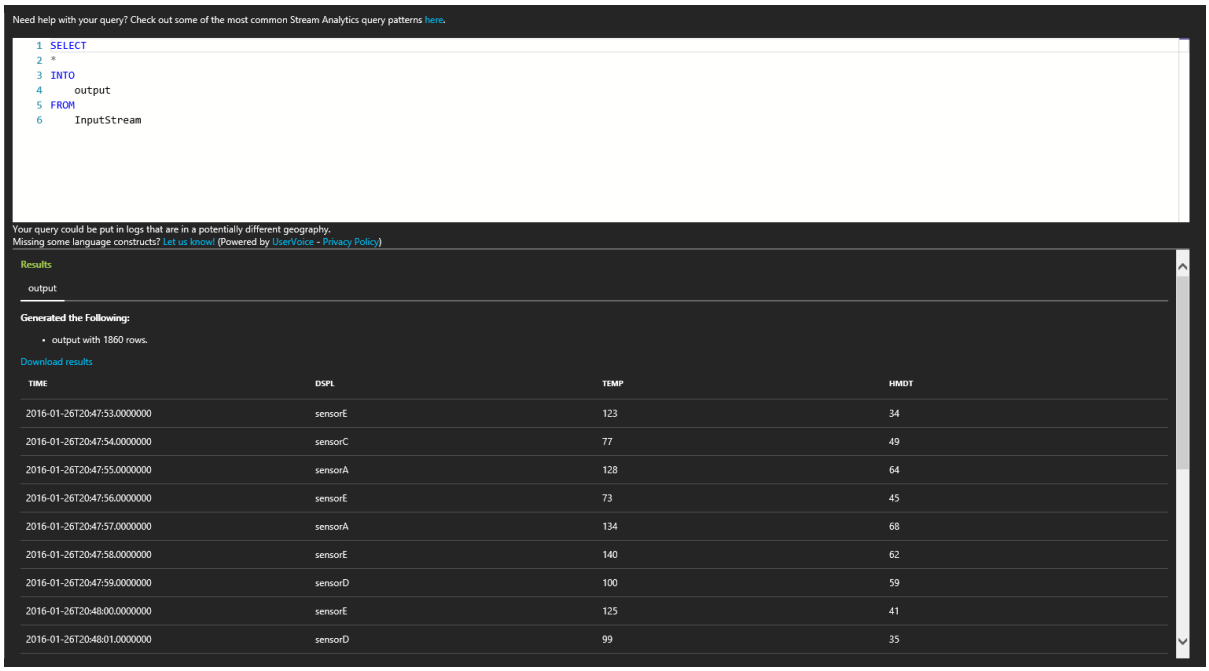
2. Click the three dots next to your input and select **Upload sample data from file** box.



3. A pane opens on the right as a result, in it select the HelloWorldASA-InputStream.json data file from your downloaded location and click **OK** at the bottom of the pane.

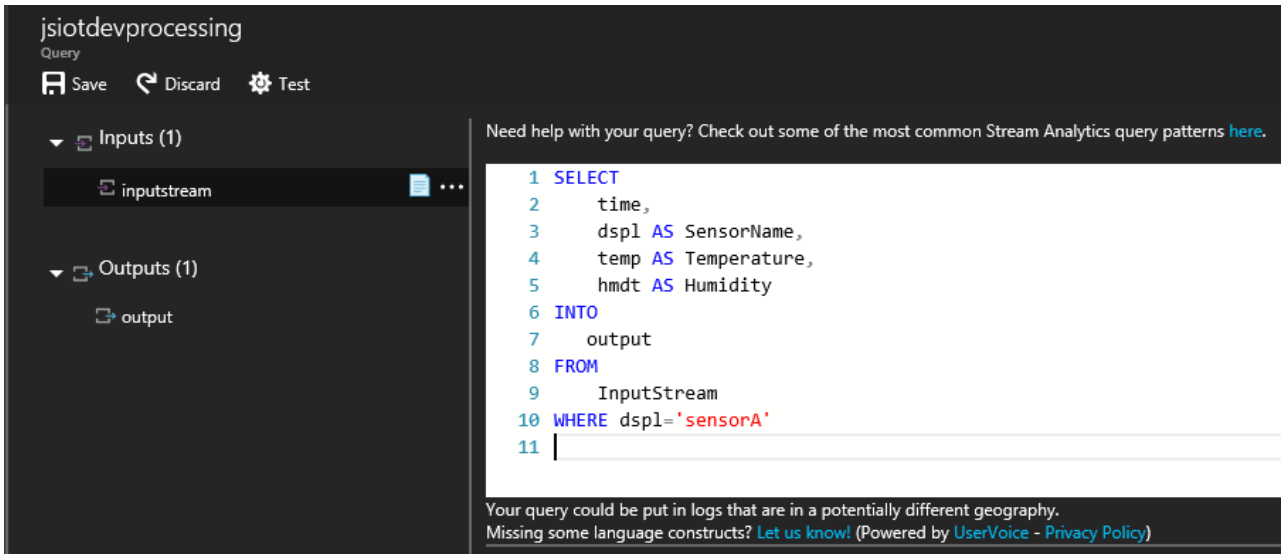


4. Then click the **Test** gear in the top left area of the window and process your test query against the sample dataset. A results window will open below your query as the processing is complete.

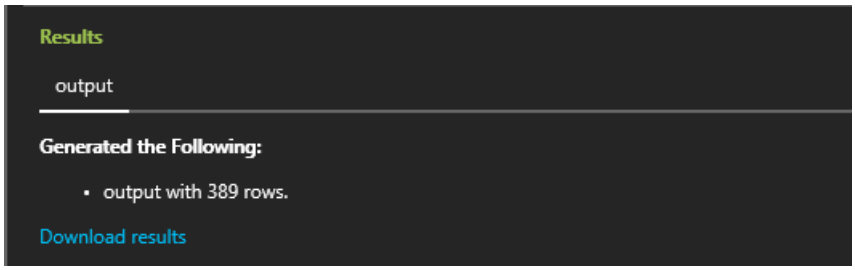


Query: Filter the data based on a condition

Let's try to filter the results based on a condition. We would like to show results for only those events that come from "sensorA." The query is in the Filtering.txt file.

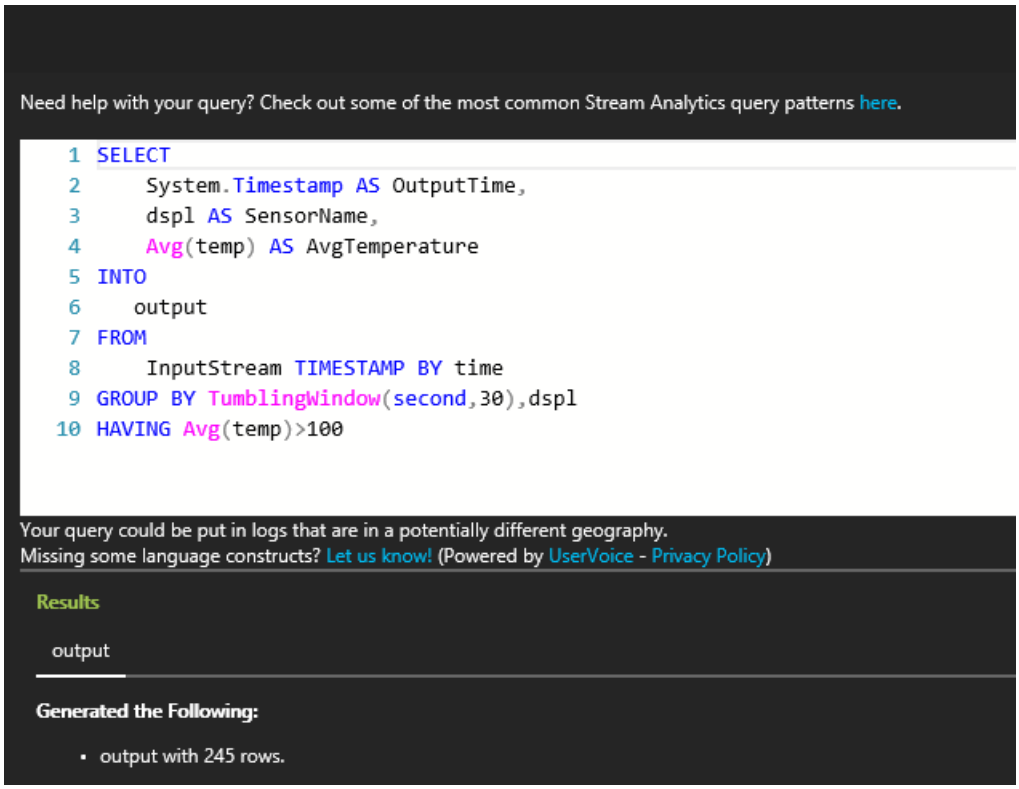


Note that the case-sensitive query compares a string value. Click the **Test** gear again to execute the query. The query should return 389 rows out of 1860 events.



Query: Alert to trigger a business workflow

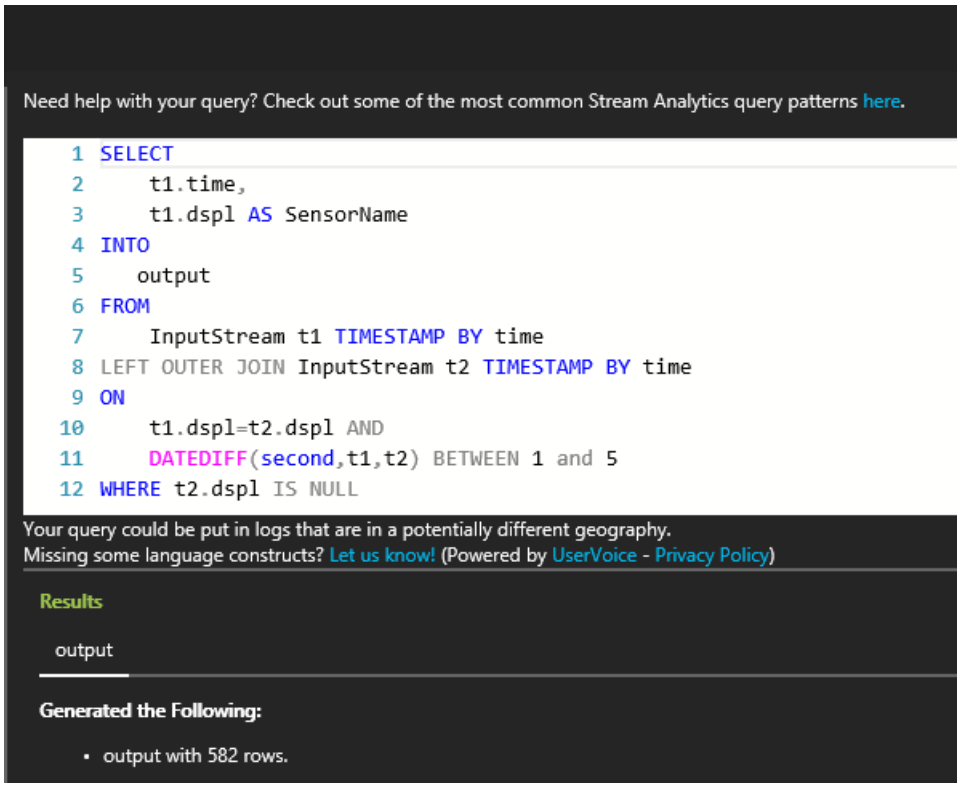
Let's make our query more detailed. For every type of sensor, we want to monitor average temperature per 30-second window and display results only if the average temperature is above 100 degrees. We will write the following query and then click **Test** to see the results. The query is in the ThresholdAlerting.txt file.



You should now see results that contain only 245 rows and names of sensors where the average temperate is greater than 100. This query groups the stream of events by **dspl**, which is the sensor name, over a **Tumbling Window** of 30 seconds. Temporal queries must state how we want time to progress. By using the **TIMESTAMP BY** clause, we have specified the **OUTPUTTIME** column to associate times with all temporal calculations. For detailed information, read the MSDN articles about Time Management and Windowing functions.

Query: Detect absence of events

How can we write a query to find a lack of input events? Let's find the last time that a sensor sent data and then did not send events for the next minute. The query is in the AbsenseOfEvent.txt file.



Here we use a **LEFT OUTER** join to the same data stream (self-join). For an **INNER** join, a result is returned only when a match is found. For a **LEFT OUTER** join, if an event from the left side of the join is unmatched, a row that

has NULL for all the columns of the right side is returned. This technique is very useful to find an absence of events. See our MSDN documentation for more information about JOIN.

Conclusion

The purpose of this tutorial is to demonstrate how to write different Stream Analytics Query Language queries and see results in the browser. However, this is just getting started. You can do so much more with Stream Analytics. Stream Analytics supports a variety of inputs and outputs and can even use functions in Azure Machine Learning to make it a robust tool for analyzing data streams. You can start to explore more about Stream Analytics by using our learning map. For more information about how to write queries, read the article about common query patterns.

Get started using Azure Stream Analytics: Real-time fraud detection

1/25/2017 • 10 min to read • Edit on GitHub

Learn how to create an end-to-end solution for real-time fraud detection with Azure Stream Analytics. Bring events into Azure Event Hubs, write Stream Analytics queries for aggregation or alerting, and send the results to an output sink to gain insights over data with real-time processing. Real-time anomaly detection for telecommunications is explained, but the example technique is equally suited for other types of fraud detection such as credit card or identity theft scenarios.

Stream Analytics is a fully managed service that provides low-latency, highly available, scalable, complex event processing over streaming data in the cloud. For more information, see Introduction to Azure Stream Analytics.

Scenario: Telecommunications and SIM fraud detection in real time

A telecommunications company has a large volume of data for incoming calls. The company needs the following from its data:

- Reduce data to a manageable amount and obtain insights about customer usage over time and across geographical regions.
- Detect SIM fraud (multiple calls from the same identity around the same time but in geographically different locations) in real time so that the company can easily respond by notifying customers or shutting down service.

Canonical Internet of Things (IoT) scenarios have a ton of telemetry or data from sensors. Customers want to aggregate the data or receive alerts about anomalies in real time.

Prerequisites

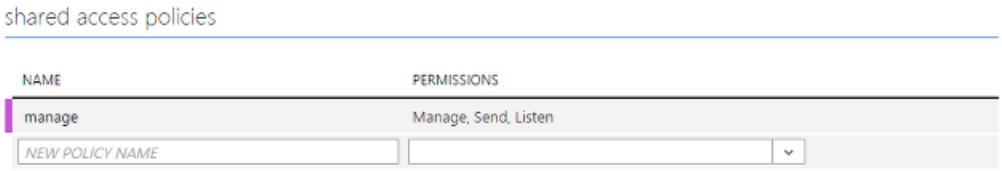
- Download TelcoGenerator.zip from the Microsoft Download Center
- Optional: Source code of the event generator from GitHub

Create Azure Event Hubs input and consumer group

The sample application will generate events and push them to an Event Hubs instance for real-time processing. Service Bus Event Hubs are the preferred method of event ingestion for Stream Analytics. You can learn more about Event Hubs in Azure Service Bus documentation.

To create an event hub:

1. In the Azure portal, click **NEW** > **APP SERVICES** > **SERVICE BUS** > **EVENT HUB** > **QUICK CREATE**. Provide a name, region, and new or existing namespace to create a new event hub.
2. As a best practice, each Stream Analytics job should read from a single event hub consumer group. We will walk you through the process of creating a consumer group later. Learn more about Consumer Groups. To create a consumer group, go to the newly created event hub, click the **CONSUMER GROUPS** tab, click **CREATE** on the bottom of the page, and then provide a name for your consumer group.
3. To grant access to the event hub, we will need to create a shared access policy. Click the **CONFIGURE** tab of your event hub.
4. Under **SHARED ACCESS POLICIES**, create a new policy that has **MANAGE** permissions.



5. Click **SAVE** at the bottom of the page.
6. Go to the **Dashboard**, click **CONNECTION INFORMATION** at the bottom of the page, and then copy and save the connection information.

Configure and start the event generator application

We have provided a client application that will generate sample incoming call metadata and push it to Event Hubs. Use the following steps to set up this application.

1. Download the TelcoGenerator.zip file, and unzip it to a directory.

   > **NOTE**
   > Windows may block the downloaded zip file. Right-click the file, and select **Properties**. If you see the "This file came from another computer and might be blocked to help protect this computer" message, select the **Unblock** box, and then click apply on the zip file.

2. Replace the Microsoft.ServiceBus.ConnectionString and EventHubName values in telcodatagen.exe.config with your event hub connection string and name.

   The connection string that you copied from the Azure portal places the name of the connection at the end. Be sure to remove ";EntityPath=" from the "add key=" field.

3. Start the application. The usage is as follows:

   telcodatagen.exe [#NumCDRsPerHour] [SIM Card Fraud Probability] [#DurationHours]

The following example will generate 1,000 events with a 20 percent probability of fraud over the course of two hours.

```
telcodatagen.exe 1000 .2 2
```

You will see records being sent to your event hub. Some key fields that we will be using in this real-time fraud detection application are defined here:

RECORD	DEFINITION
CallrecTime	Timestamp for the call start time.
SwitchNum	Telephone switch used to connect the call.
CallingNum	Phone number of the caller.
CallingIMSI	International Mobile Subscriber Identity (IMSI). Unique identifier of the caller.
CalledNum	Phone number of the call recipient.

RECORD	DEFINITION
CalledIMSI	International Mobile Subscriber Identity (IMSI). Unique identifier of the call recipient.

Create a Stream Analytics job

Now that we have a stream of telecommunications events, we can set up a Stream Analytics job to analyze these events in real time.

Provision a Stream Analytics job

1. In the Azure portal, click **NEW** > **DATA SERVICES** > **STREAM ANALYTICS** > **QUICK CREATE**.
2. Specify the following values, and then click **CREATE STREAM ANALYTICS JOB**:

   - **JOB NAME**: Enter a job name.
   - **REGION**: Select the region where you want to run the job. Consider placing the job and the event hub in the same region to ensure better performance and to ensure that you will not be paying to transfer data between regions.
   - **STORAGE ACCOUNT**: Choose the Azure storage account that you would like to use to store monitoring data for all Stream Analytics jobs that run within this region. You have the option to choose an existing storage account or create a new one.
3. Click **STREAM ANALYTICS** in the left pane to list the Stream Analytics jobs.

   The new job will be shown with a status of **CREATED**. Notice that the **START** button on the bottom of the page is disabled. You must configure the job input, output, and query before you can start the job.

Specify job input

1. In your Stream Analytics job, click **INPUTS** at the top of the page, and then click **ADD INPUT**. The dialog box that opens will walk you through several steps to set up your input.
2. Click **DATA STREAM**, and then click the right button.
3. Click **EVENT HUB**, and then click the right button.
4. Type or select the following values on the third page:

   - **INPUT ALIAS**: Enter a friendly name, such as *CallStream*, for this job. Note that you will be using this name in the query later.
   - **EVENT HUB**: If the event hub that you created is in the same subscription as the Stream Analytics job, select the namespace that the event hub is in.

     If your event hub is in a different subscription, select **Use Event Hub from Another Subscription**, and then manually enter information for **SERVICE BUS NAMESPACE**, **EVENT HUB NAME**, **EVENT HUB POLICY NAME**, **EVENT HUB POLICY KEY**, and **EVENT HUB PARTITION COUNT**.

   - **EVENT HUB NAME**: Select the name of the event hub.
   - **EVENT HUB POLICY NAME**: Select the event hub policy that you created earlier in this tutorial.
   - **EVENT HUB CONSUMER GROUP**: Type the name of the consumer group that you created earlier in this tutorial.
5. Click the right button.
6. Specify the following values:

   - **EVENT SERIALIZER FORMAT**: JSON

- **ENCODING**: UTF8

7. Click the **CHECK** button to add this source and to verify that Stream Analytics can successfully connect to the event hub.

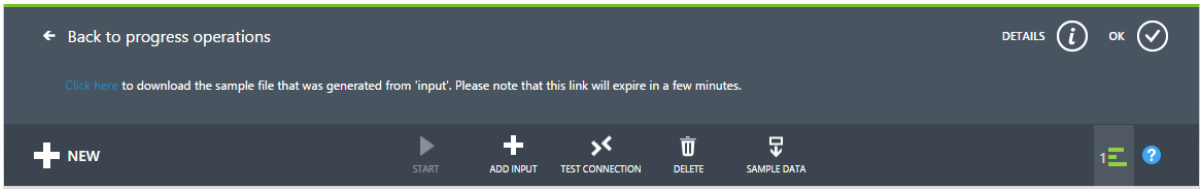
Specify job query

Stream Analytics supports a simple, declarative query model that describes transformations for real-time processing. To learn more about the language, see the Azure Stream Analytics Query Language Reference. This tutorial will help you author and test several queries over your real-time stream of call data.

Optional: Sample input data

To validate your query against actual job data, you can use the **SAMPLE DATA** feature to extract events from your stream and create a .JSON file of the events for testing. The following steps show how to do this. We have also provided a sample telco.json file for testing purposes.

1. Select your event hub input, and then click **SAMPLE DATA** at the bottom of the page.
2. In the dialog box that opens, specify a **START TIME** to start collecting data and a **DURATION** for how much additional data to consume.
3. Click the **CHECK** button to start sampling data from the input. It can take a minute or two for the data file to be produced. When the process is finished, click **DETAILS**, download the generated .JSON file, and save it.



Pass-through query

If you want to archive every event, you can use a pass-through query to read all the fields in the payload of the event or message. To start, do a simple pass-through query that projects all the fields in an event.

1. Click **QUERY** from the top of the Stream Analytics job page.
2. Add the following to the code editor:

   ```
   SELECT * FROM CallStream
   ```

   > **IMPORTANT**
   > Make sure that the name of the input source matches the name of the input that you specified earlier.

3. Click **Test** under the query editor.
4. Supply a test file. Use one that you created using the previous steps, or use telco.json.
5. Click the **CHECK** button, and see the results displayed below the query definition.

Output

RECORDTYPE	SYSTEMIDEN...	FILENUM	SWITCHNUM	CALLINGNUM	CALLINGIMSI	CALLEDNUM	CALLEDIMSI	DATES
MO	d0	3	US	123459531	466921602131...	345626629	466922002560...	20150415
MO	d0	8	UK	012333646	466923101048...	345604805	466923200779...	20150415
MO	d0	9	China	123470312	466921402237...	789049921		20150415
MO	d0	11	Germany	567828491	466921602343...	789051039	466922200432...	20150415
MO	d0	14	US	678900052	466922201102...	456752187	466921402237...	20150415

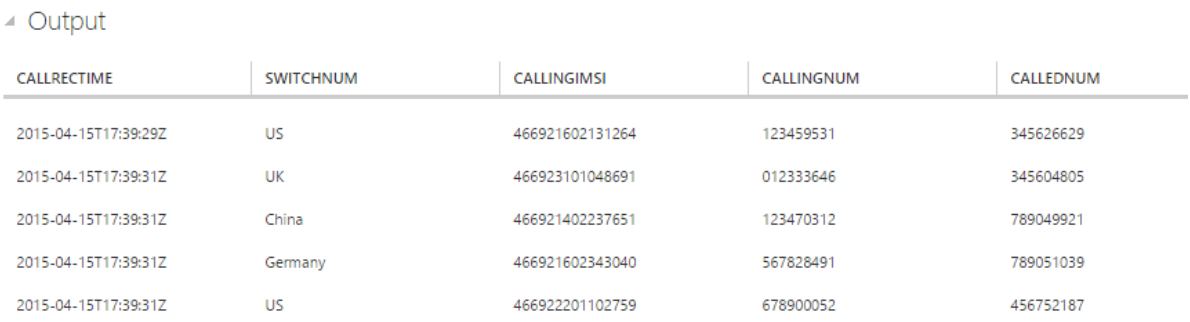
Column projection

We'll now reduce the returned fields to a smaller set.

1. Change the query in the code editor to:

   SELECT CallRecTime, SwitchNum, CallingIMSI, CallingNum, CalledNum FROM CallStream

2. Click **Rerun** under the query editor to see the results of the query.



Output

CALLRECTIME	SWITCHNUM	CALLINGIMSI	CALLINGNUM	CALLEDNUM
2015-04-15T17:39:29Z	US	466921602131264	123459531	345626629
2015-04-15T17:39:31Z	UK	466923101048691	012333646	345604805
2015-04-15T17:39:31Z	China	466921402237651	123470312	789049921
2015-04-15T17:39:31Z	Germany	466921602343040	567828491	789051039
2015-04-15T17:39:31Z	US	466922201102759	678900052	456752187

Count of incoming calls by region: Tumbling window with aggregation

To compare the number of incoming calls per region, we'll use a TumblingWindow to get the count of incoming calls grouped by **SwitchNum** every five seconds.

1. Change the query in the code editor to:

   SELECT System.Timestamp as WindowEnd, SwitchNum, COUNT(*) as CallCount FROM CallStream TIMESTAMP BY CallRecTime GROUP BY TUMBLINGWINDOW(s, 5), SwitchNum

   This query uses the **Timestamp By** keyword to specify a timestamp field in the payload to be used in the temporal computation. If this field wasn't specified, the windowing operation would be performed by using the time that each event arrived at the event hub. See "Arrival Time Vs Application Time" in the Stream Analytics Query Language Reference.

   Note that you can access a timestamp for the end of each window by using the **System.Timestamp** property.

2. Click **Rerun** under the query editor to see the results of the query.

Output

WINDOWEND	SWITCHNUM	CALLCOUNT
2015-04-14T01:20:05.000Z	Australia	1
2015-04-14T01:20:05.000Z	China	1
2015-04-14T01:20:05.000Z	Germany	1
2015-04-14T01:20:05.000Z	UK	2
2015-04-14T01:20:10.000Z	China	1

SIM fraud detection with a Self-Join

To identify potentially fraudulent usage, we'll look for calls that originate from the same user but in different locations in less than 5 seconds. We join the stream of call events with itself to check for these cases.

1. Change the query in the code editor to:

   SELECT System.Timestamp as Time, CS1.CallingIMSI, CS1.CallingNum as CallingNum1, CS2.CallingNum as CallingNum2, CS1.SwitchNum as Switch1, CS2.SwitchNum as Switch2 FROM CallStream CS1 TIMESTAMP BY CallRecTime JOIN CallStream CS2 TIMESTAMP BY CallRecTime ON CS1.CallingIMSI = CS2.CallingIMSI

AND DATEDIFF(ss, CS1, CS2) BETWEEN 1 AND 5 WHERE CS1.SwitchNum != CS2.SwitchNum

2. Click **Rerun** under the query editor to see the results of the query.


Output

TIME	CALLINGIMSI	CALLINGNUM1	CALLINGNUM2	SWITCH1	SWITCH2
2015-04-15T18:05:58.000Z	466921602131264	234560397	678957514	US	Australia
2015-04-15T17:40:59.000Z	466922200432822	678933469	789097907	Australia	Germany
2015-04-15T17:40:58.000Z	466921602343040	234523973	345692262	US	UK
2015-04-15T17:40:58.000Z	466920401237309	234556280	678963520	US	Australia
2015-04-15T17:40:56.000Z	466920401237309	234556280	345669096	US	Australia

Create output sink

Now that we have defined an event stream, an event hub input to ingest events, and a query to perform a transformation over the stream, the last step is to define an output sink for the job. We'll write events for fraudulent behavior to Azure Blob storage.

Use the following steps to create a container for Blob storage if you don't already have one.

1. Use an existing storage account or create a new storage account by clicking **NEW > DATA SERVICES > STORAGE > QUICK CREATE**, and follow the instructions.
2. Select the storage account, click **CONTAINERS** at the top of the page, and then click **ADD**.
3. Specify a **NAME** for your container, and set its **ACCESS** to **Public Blob**.

Specify job output

1. In your Stream Analytics job, click **OUTPUT** at the top of the page, and then click **ADD OUTPUT**. The dialog box that opens will walk you through several steps to set up your output.
2. Click **BLOB STORAGE**, and then click the right button.
3. Type or select the following values on the third page:

   - **OUTPUT ALIAS**: Enter a friendly name for this job output.
   - **SUBSCRIPTION**: If the Blob storage that you created is in the same subscription as the Stream Analytics job, click **Use Storage Account from Current Subscription**. If your storage is in a different subscription, click **Use Storage Account from Another Subscription**, and manually enter information for **STORAGE ACCOUNT**, **STORAGE ACCOUNT KEY**, and **CONTAINER**.
   - **STORAGE ACCOUNT**: Select the name of the storage account.
   - **CONTAINER**: Select the name of the container.
   - **FILENAME PREFIX**: Type a file prefix to use when writing blob output.
4. Click the right button.
5. Specify the following values:

   - **EVENT SERIALIZER FORMAT**: JSON
   - **ENCODING**: UTF8
6. Click the **CHECK** button to add this source and to verify that Stream Analytics can successfully connect to the storage account.

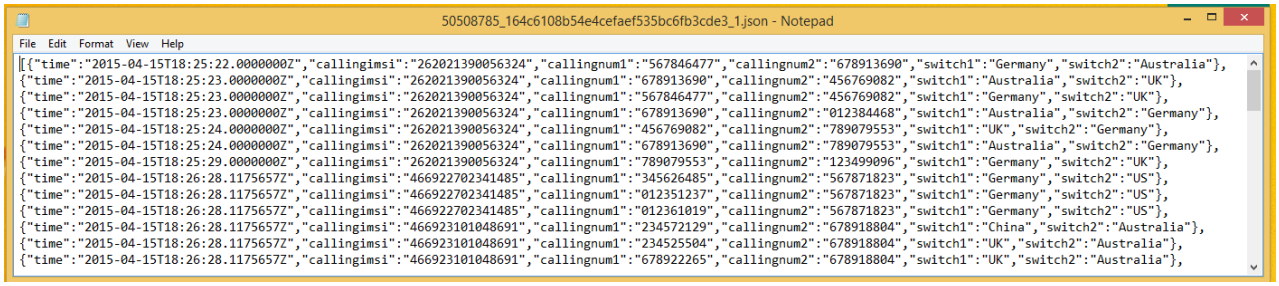
Start job for real-time processing

Because a job input, query, and output have all been specified, we are ready to start the Stream Analytics job for real-time fraud detection.

1. From the job **DASHBOARD**, click **START** at the bottom of the page.
2. In the dialog box that opens, click **JOB START TIME**, and then click the **CHECK** button on the bottom of the dialog box. The job status will change to **Starting** and will shortly change to **Running**.

View fraud detection output

Use a tool like Azure Storage Explorer or Azure Explorer to view fraudulent events as they are written to your output in real time.



Get support

For further assistance, try our Azure Stream Analytics forum.

Next steps

- Introduction to Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Build an IoT solution by using Stream Analytics

1/25/2017 • 16 min to read • Edit on GitHub

Introduction

In this tutorial, you will learn how to use Azure Stream Analytics to get real-time insights from your data. Developers can easily combine streams of data, such as click-streams, logs, and device-generated events, with historical records or reference data to derive business insights. As a fully managed, real-time stream computation service that's hosted in Microsoft Azure, Azure Stream Analytics provides built-in resiliency, low latency, and scalability to get you up and running in minutes.

After completing this tutorial, you will be able to:

- Familiarize yourself with the Azure Stream Analytics portal.
- Configure and deploy a streaming job.
- Articulate real-world problems and solve them by using the Stream Analytics query language.
- Develop streaming solutions for your customers by using Stream Analytics with confidence.
- Use the monitoring and logging experience to troubleshoot issues.

Prerequisites

You will need the following prerequisites to complete this tutorial:

- The latest version of Azure PowerShell
- Visual Studio 2015 or the free Visual Studio Community
- An Azure subscription
- Administrative privileges on the computer
- Download of TollApp.zip from the Microsoft Download Center
- Optional: Source code for the TollApp event generator in GitHub

Scenario introduction: “Hello, Toll!”

A toll station is a common phenomenon. You encounter them on many expressways, bridges, and tunnels across the world. Each toll station has multiple toll booths. At manual booths, you stop to pay the toll to an attendant. At automated booths, a sensor on top of each booth scans an RFID card that's affixed to the windshield of your vehicle as you pass the toll booth. It is easy to visualize the passage of vehicles through these toll stations as an event stream over which interesting operations can be performed.

Incoming data

This tutorial works with two streams of data. Sensors installed in the entrance and exit of the toll stations produce the first stream. The second stream is a static lookup dataset that has vehicle registration data.

Entry data stream

The entry data stream contains information about cars as they enter toll stations.

TOLLID	ENTRYTIME	LICENSEPLATE	STATE	MAKE	MODEL	VEHICLETYPE	VEHICLEWEIGHT	TOLL	TAG
1	2014-09-10 12:01:00.000	JNB 7001	NY	Honda	CRV	1	0	7	
1	2014-09-10 12:02:00.000	YXZ 1001	NY	Toyota	Camry	1	0	4	123456789
3	2014-09-10 12:02:00.000	ABC 1004	CT	Ford	Taurus	1	0	5	456789123
2	2014-09-10 12:03:00.000	XYZ 1003	CT	Toyota	Corolla	1	0	4	
1	2014-09-10 12:03:00.000	BNJ 1007	NY	Honda	CRV	1	0	5	789123456
2	2014-09-10 12:05:00.000	CDE 1007	NJ	Toyota	4x4	1	0	6	321987654

Here is a short description of the columns:

COLUMN	DESCRIPTION
TollID	The toll booth ID that uniquely identifies a toll booth
EntryTime	The date and time of entry of the vehicle to the toll booth in UTC
LicensePlate	The license plate number of the vehicle
State	A state in United States
Make	The manufacturer of the automobile
Model	The model number of the automobile
VehicleType	Either 1 for passenger vehicles or 2 for commercial vehicles
WeightType	Vehicle weight in tons; 0 for passenger vehicles
Toll	The toll value in USD
Tag	The e-Tag on the automobile that automates payment; blank where the payment was done manually

Exit data stream

The exit data stream contains information about cars leaving the toll station.

TOLLID	EXITTIME	LICENSEPLATE
1	2014-09-10T12:03:00.0000000Z	JNB 7001
1	2014-09-10T12:03:00.0000000Z	YXZ 1001
3	2014-09-10T12:04:00.0000000Z	ABC 1004
2	2014-09-10T12:07:00.0000000Z	XYZ 1003
1	2014-09-10T12:08:00.0000000Z	BNJ 1007
2	2014-09-10T12:07:00.0000000Z	CDE 1007

Here is a short description of the columns:

COLUMN	DESCRIPTION
TollID	The toll booth ID that uniquely identifies a toll booth
ExitTime	The date and time of exit of the vehicle from toll booth in UTC
LicensePlate	The license plate number of the vehicle

Commercial vehicle registration data

The tutorial uses a static snapshot of a commercial vehicle registration database.

LICENSEPLATE	REGISTRATIONID	EXPIRED
SVT 6023	285429838	1
XLZ 3463	362715656	0
BAC 1005	876133137	1
RIV 8632	992711956	0
SNY 7188	592133890	0
ELH 9896	678427724	1

Here is a short description of the columns:

COLUMN	DESCRIPTION
LicensePlate	The license plate number of the vehicle
RegistrationId	The vehicle's registration ID
Expired	The registration status of the vehicle: 0 if vehicle registration is active, 1 if registration is expired

Set up the environment for Azure Stream Analytics

To complete this tutorial, you need a Microsoft Azure subscription. Microsoft offers free trial for Microsoft Azure services.

If you do not have an Azure account, you can request a free trial version.

> **NOTE**
> To sign up for a free trial, you need a mobile device that can receive text messages and a valid credit card.

Be sure to follow the steps in the "Clean up your Azure account" section at the end of this article so that you can make the best use of your $200 free Azure credit.

Provision Azure resources required for the tutorial

This tutorial requires two event hubs to receive *entry* and *exit* data streams. Azure SQL Database outputs the results of the Stream Analytics jobs. Azure Storage stores reference data about vehicle registrations.

You can use the Setup.ps1 script in the TollApp folder on GitHub to create all required resources. In the interest of time, we recommend that you run it. If you would like to learn more about how to configure these resources in the Azure portal, refer to the "Configuring tutorial resources in Azure portal" appendix.

Download and save the supporting TollApp folder and files.

Open a **Microsoft Azure PowerShell** window *as an administrator*. If you do not yet have Azure PowerShell, follow the instructions in Install and configure Azure PowerShell to install it.

Because Windows automatically blocks .ps1, .dll, and .exe files, you need to set the execution policy before you run

the script. Make sure the Azure PowerShell window is running *as an administrator*. Run **Set-ExecutionPolicy unrestricted**. When prompted, type **Y**.

```
PS C:\> Set-ExecutionPolicy unrestricted

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
http://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): Y
PS C:\>
```

Run **Get-ExecutionPolicy** to make sure that the command worked.

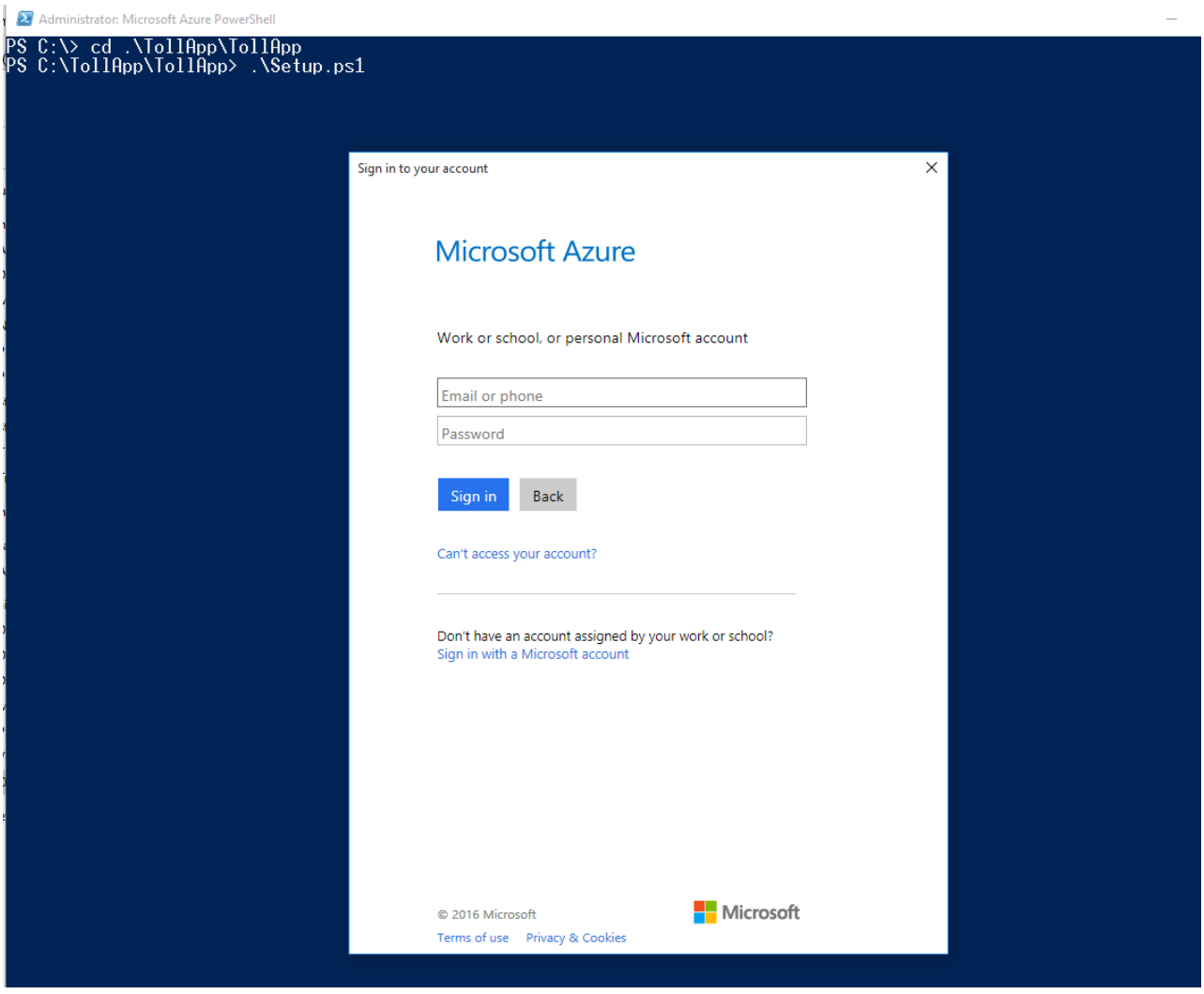
```
PS C:\> Get-ExecutionPolicy
Unrestricted
PS C:\>
```

Go to the directory that has the scripts and generator application.

```
PS C:\> cd .\TollApp\TollApp
PS C:\TollApp\TollApp>
```

Type **.\Setup.ps1** to set up your Azure account, create and configure all required resources, and start to generate events. The script randomly picks up a region to create your resources. To explicitly specify a region, you can pass the **-location** parameter as in the following example:

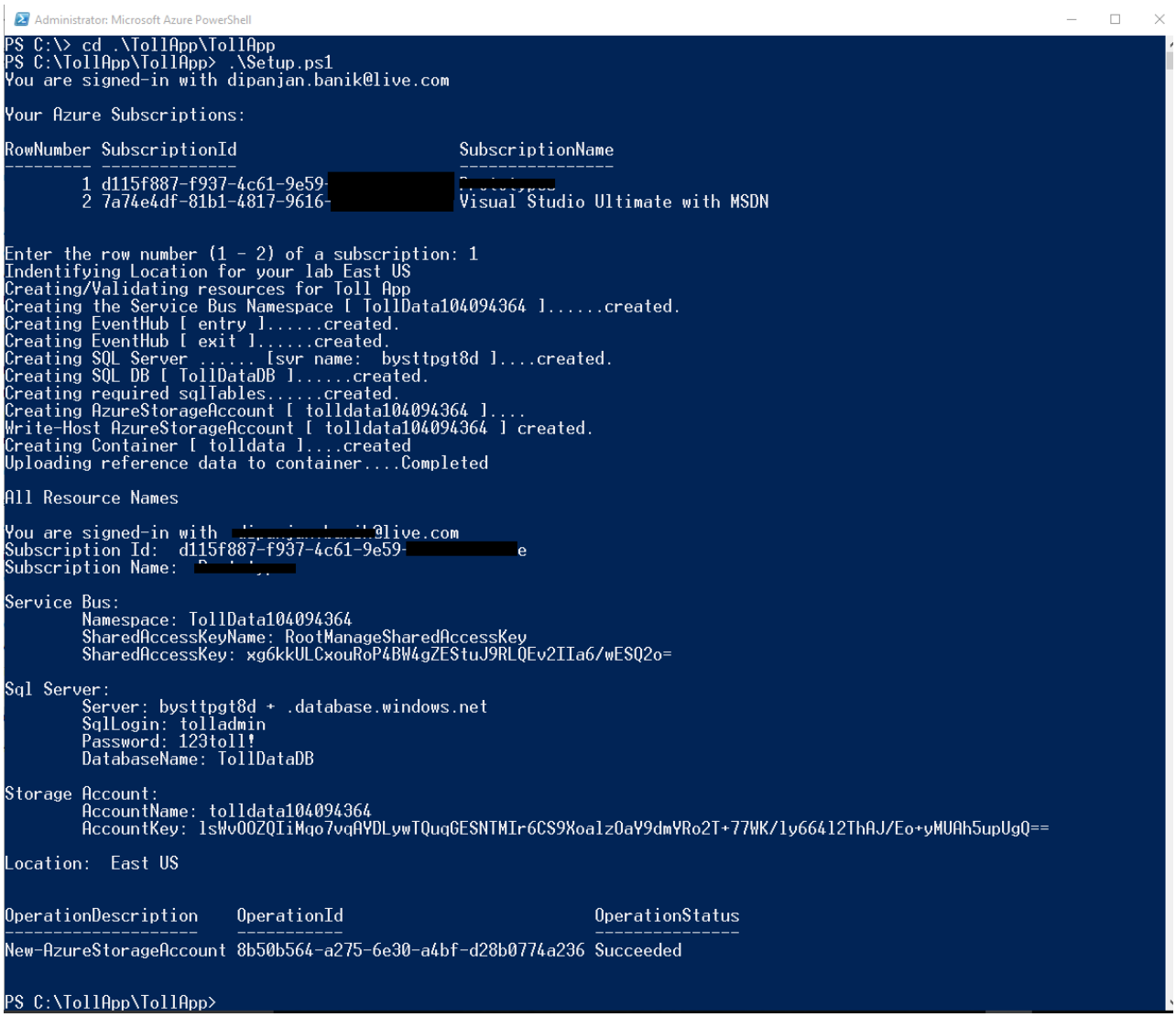
.\Setup.ps1 -location "Central US"



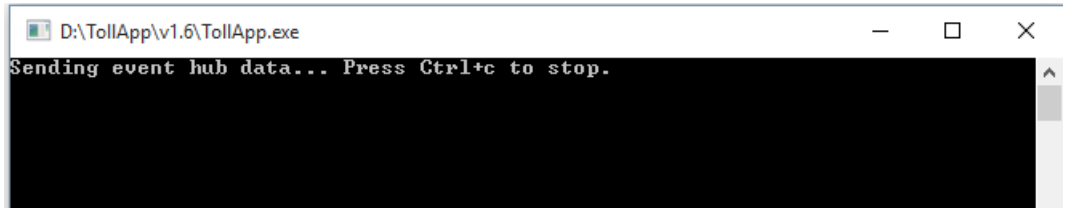
The script opens the **Sign In** page for Microsoft Azure. Enter your account credentials.

> **NOTE**
>
> If your account has access to multiple subscriptions, you will be asked to enter the subscription name that you want to use for the tutorial.

The script can take several minutes to run. After it finishes, the output should look like the following screenshot.



You will also see another window that's similar to the following screenshot. This application is sending events to Azure Event Hubs, which is required to run the tutorial. So, do not stop the application or close this window until you finish the tutorial.

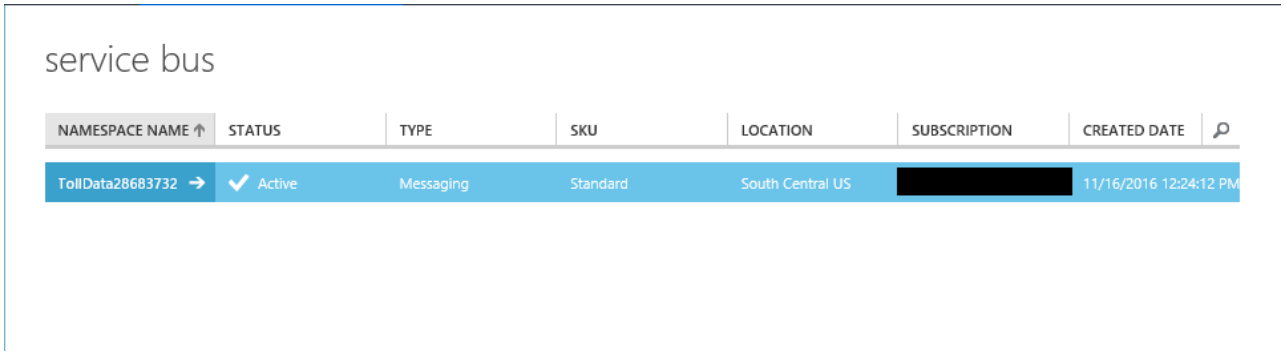


You should be able to see your resources in Azure portal now. Go to https://portal.azure.com, and sign in with your account credentials. Note that currently some functionality utilizes the classic portal. These steps will be clearly indicated.

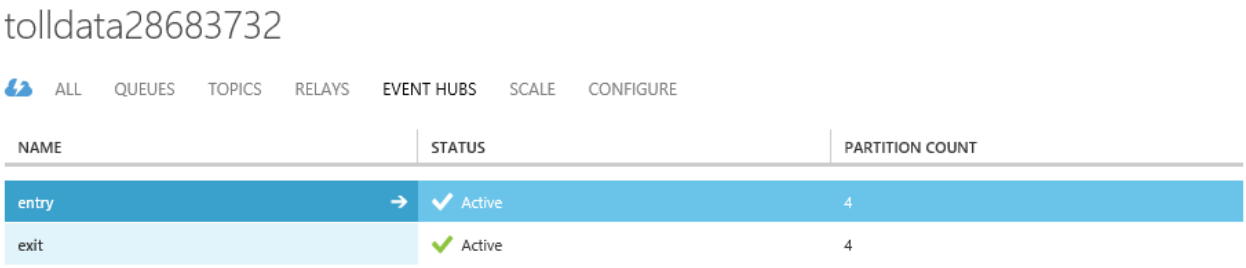
Azure Event Hubs

In the Azure portal, click **More services** on the bottom of the left management pane. Type **Event hubs** in the field provided and click **Event hubs**. This launches a new browser window to display the **SERVICE BUS** area in the

classic portal. Here you can see the Event Hub created by the Setup.ps1 script.



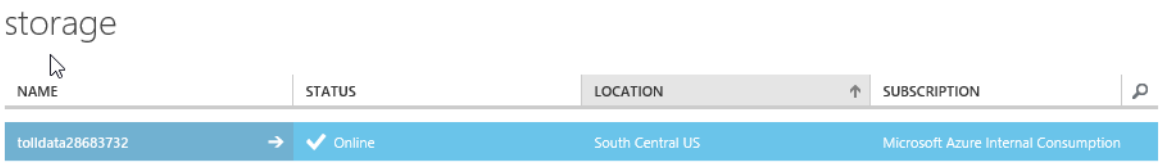
Click the one that starts with *tolldata*. Click the **EVENT HUBS** tab. You will see two event hubs named *entry* and *exit* created in this namespace.



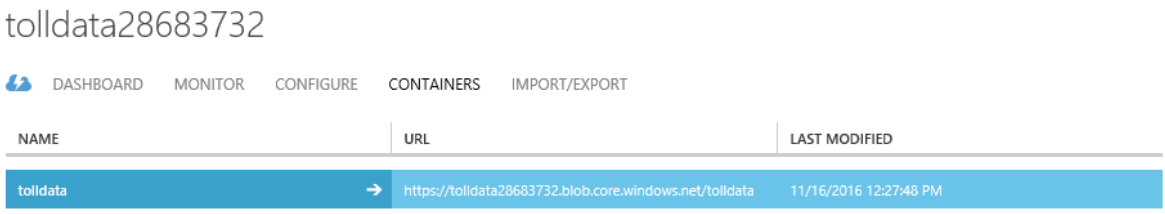
Azure Storage container

1. Go back to the tab in your browser open to Azure portal. Click **STORAGE** on the left side of the Azure portal to see the Azure Storage container that's used in the tutorial.

2. Click the one that start with *tolldata*. Click the **CONTAINERS** tab to see the created container.

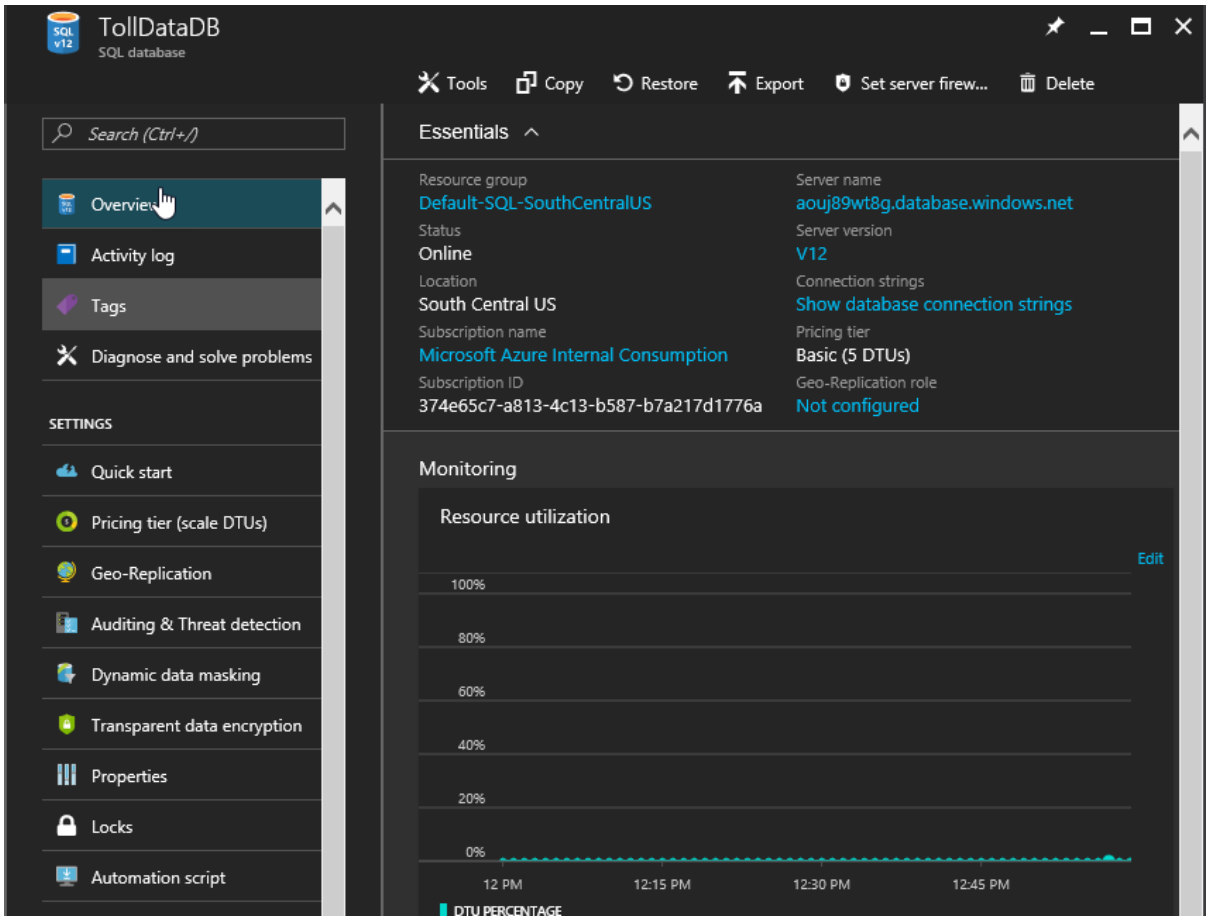


3. Click the **tolldata** container to see the uploaded JSON file that has vehicle registration data.

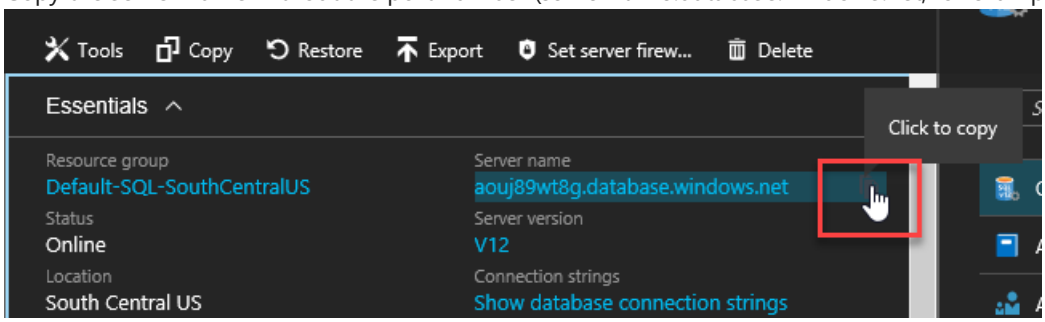


Azure SQL Database

1. Go back to the Azure portal on the first tab that was opened in the browser. Click **SQL DATABASES** on the left side of the Azure portal to see the SQL database that will be used in the tutorial and click **tolldatadb**.



2. Copy the server name without the port number (*servername*.database.windows.net, for example).

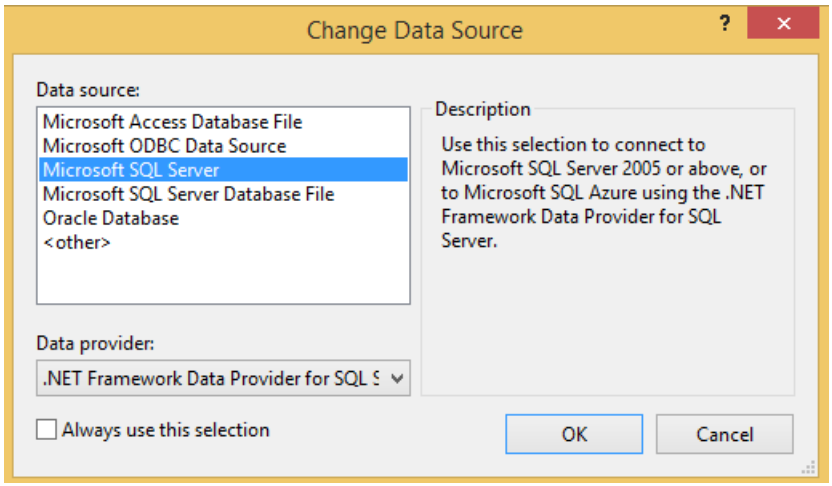


Connect to the database from Visual Studio

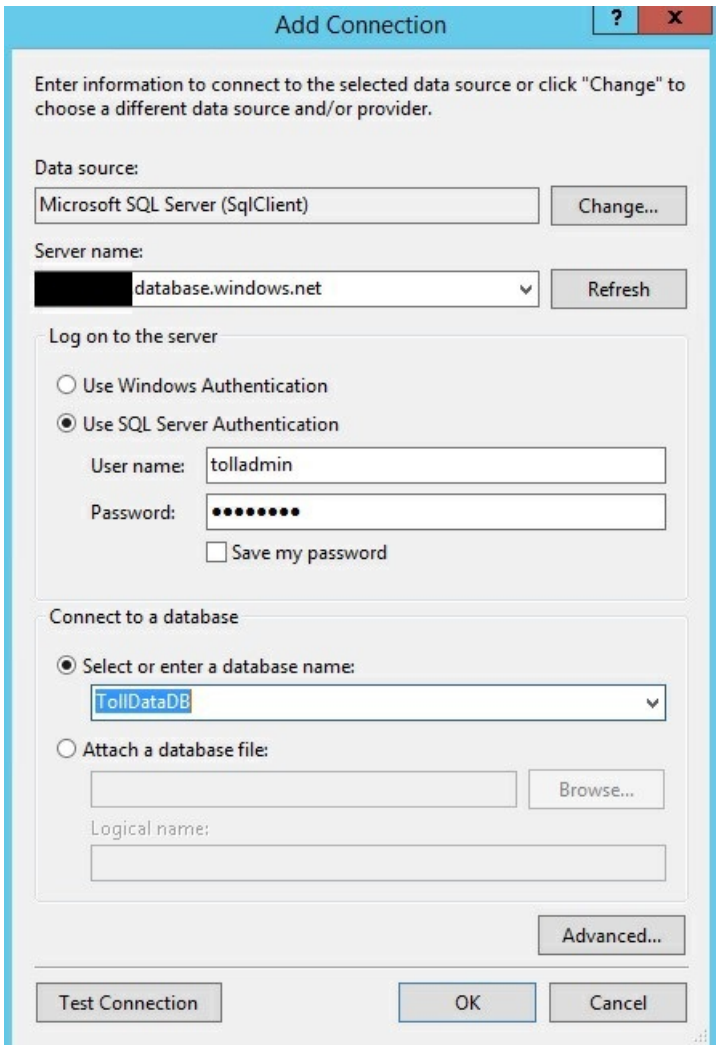
Use Visual Studio to access query results in the output database.

Connect to the SQL database (the destination) from Visual Studio:

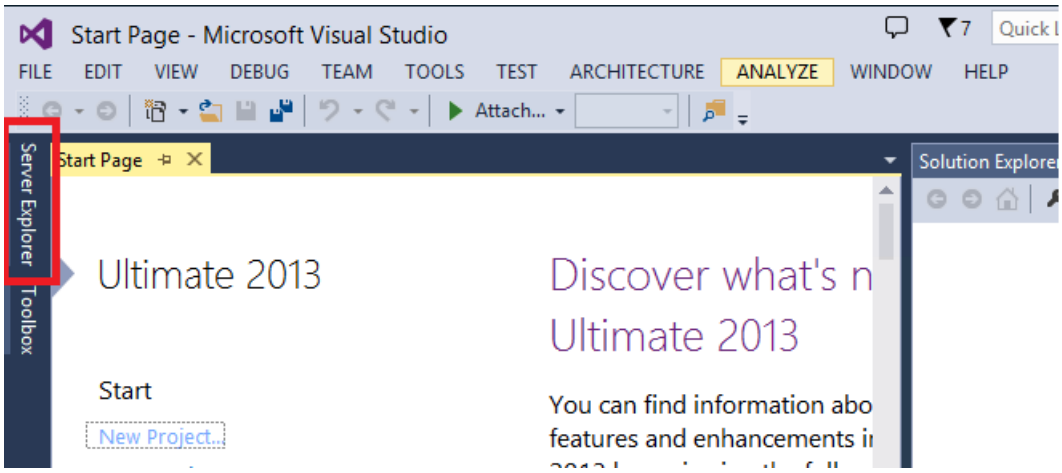
1. Open Visual Studio, and then click **Tools** > **Connect to Database**.
2. If asked, click **Microsoft SQL Server** as a data source.



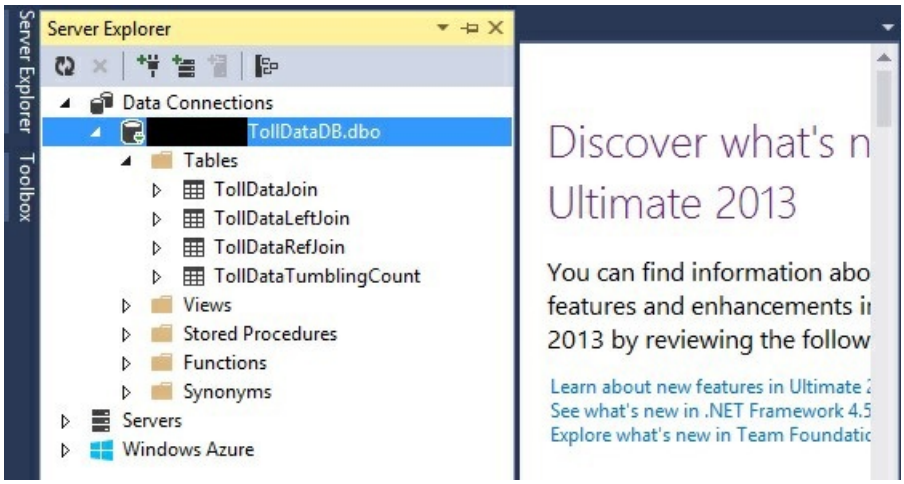
3. In the **Server name** field, paste the name that you copied in the previous section from the Azure portal (that is, *servername*.database.windows.net).
4. Click **Use SQL Server Authentication**.
5. Enter **tolladmin** in the **User name** field and **123toll!** in the **Password** field.
6. Click **Select or enter a database name**, and select **TollDataDB** as the database.



7. Click **OK**.
8. Open Server Explorer.



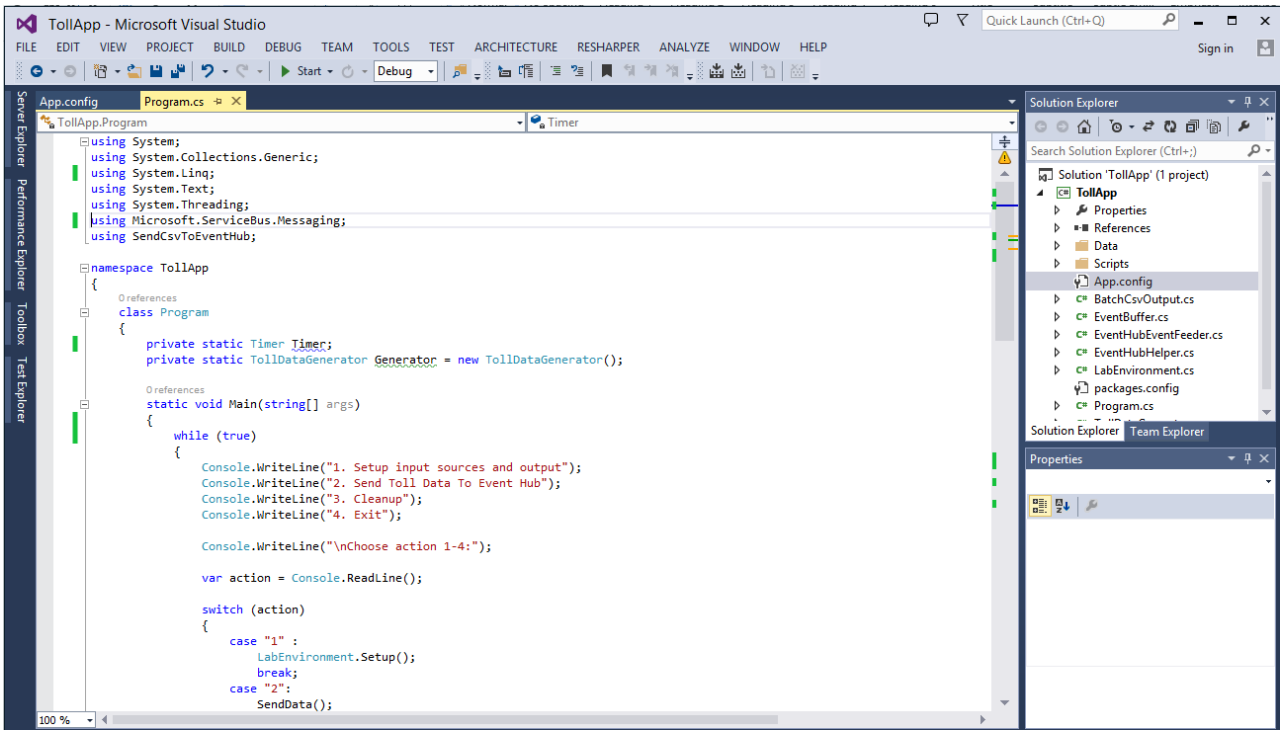
9. See four tables in the TollDataDB database.



Event generator: TollApp sample project

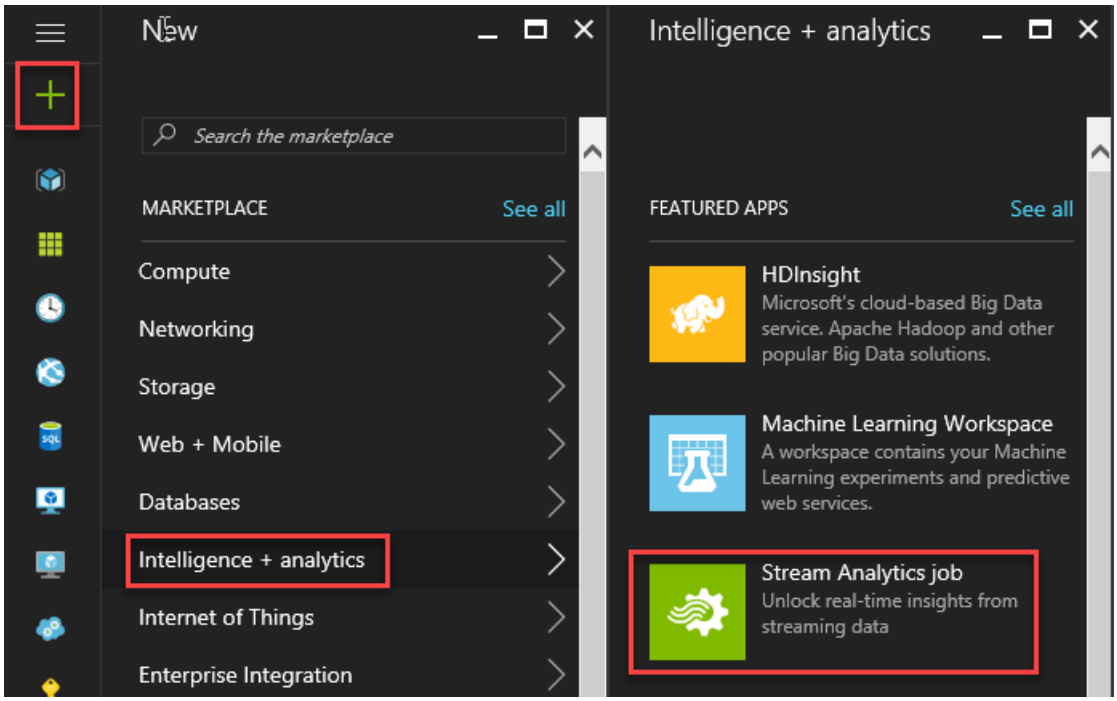
The PowerShell script automatically starts to send events by using the TollApp sample application program. You don't need to perform any additional steps.

However, if you are interested in implementation details, you can find the source code of the TollApp application in GitHub samples/TollApp.

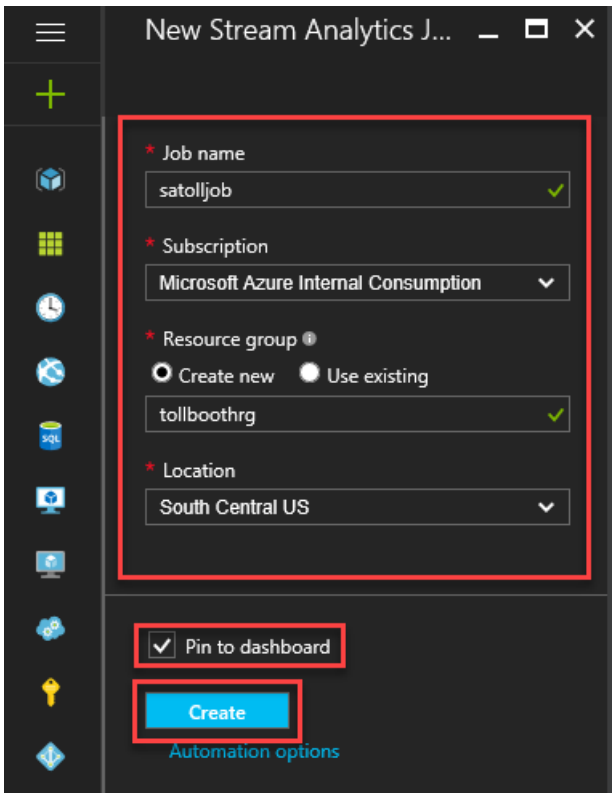


Create a Stream Analytics job

1. In the Azure portal, click the green plus sign in the top-left corner of the page to create a new Stream Analytics job. Select **Intelligence + Analytics** and then click **Stream Analytics job**.

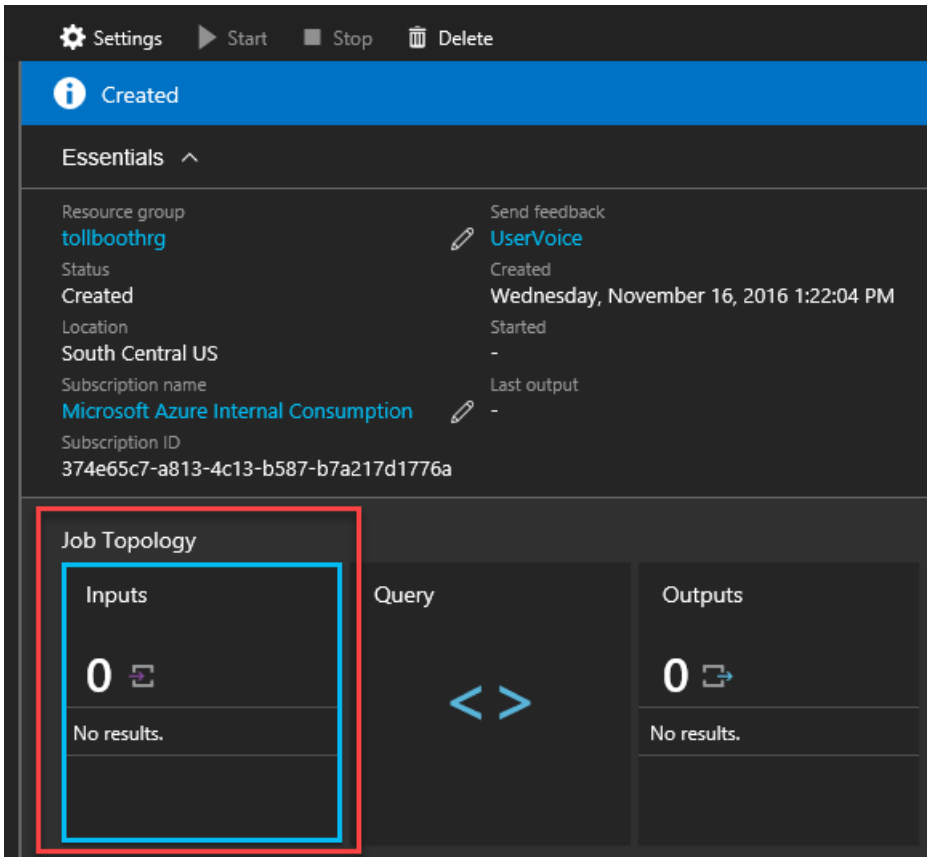


2. Provide a job name, validate the subscription is correct and then create a new Resource group in the same region as the Event hub storage (default is South Central US for the script).
3. Click **Pin to dashboard** and then **CREATE** at the bottom of the page.

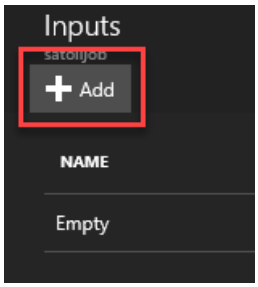


Define input sources

1. The job will create and open the job page. Or you can click the created analytics job on the portal dashboard.
2. Click the **INPUTS** tab to define the source data.

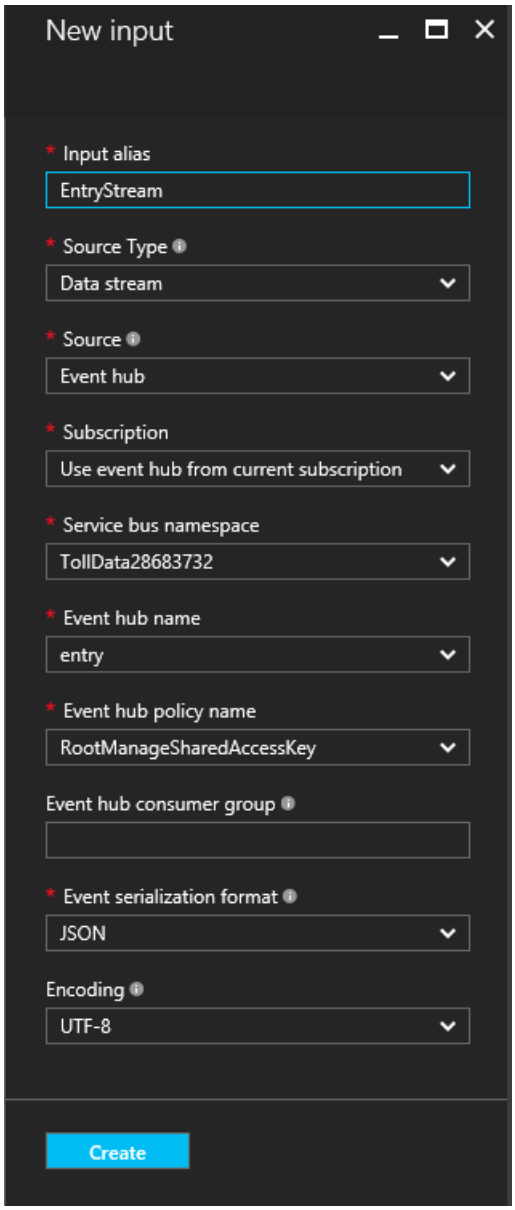


3. Click **ADD AN INPUT**.



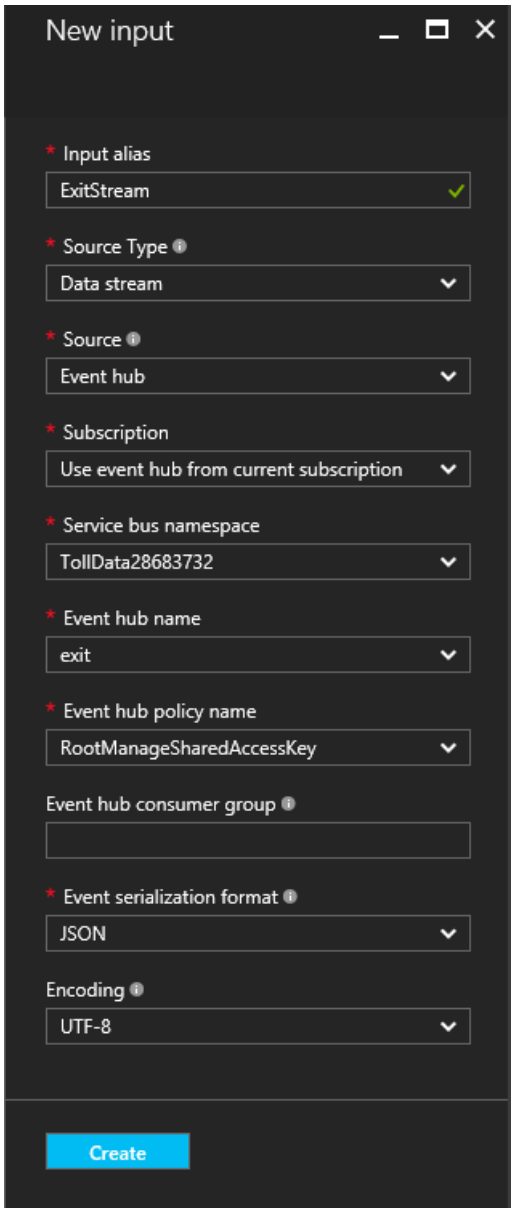
4. Enter **EntryStream** as **INPUT ALIAS**.
5. Source Type is **Data Stream**
6. Source is **Event hub**.
7. **Service bus namespace** should be the TollData one in the drop down.
8. **Event hub name** should be set to **entry**.
9. **Event hub policy name*is **RootManageSharedAccessKey** (the default value).
10. Select **JSON** for **EVENT SERIALIZATION FORMAT** and **UTF8** for **ENCODING**.

    Your settings will look like:

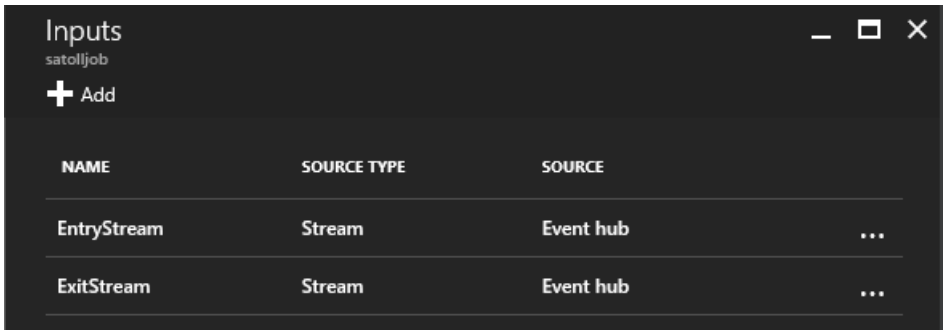


11. Click **Create** at the bottom of the page to finish the wizard.

    Now that you've created the entry stream, you will follow the same steps to create the exit stream. Be sure to enter values as on the following screenshot.



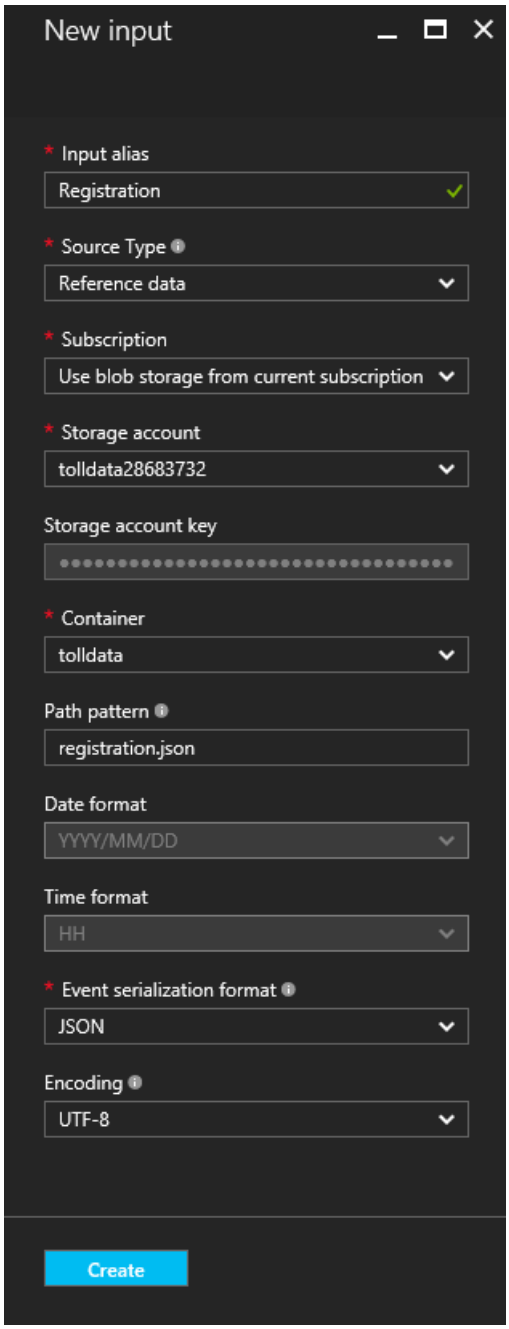
You have defined two input streams:



Next, you will add reference data input for the blob file that contains car registration data.

12. Click **ADD**, and then follow the same process for the stream inputs but select **REFERENCE DATA** instead of **Data Stream** and the **Input Alias** is **Registration**.

13. storage account that starts with **tolldata**. The container name should be **tolldata**, and the **PATH PATTERN** should be **registration.json**. This file name is case sensitive and should be **lowercase**.

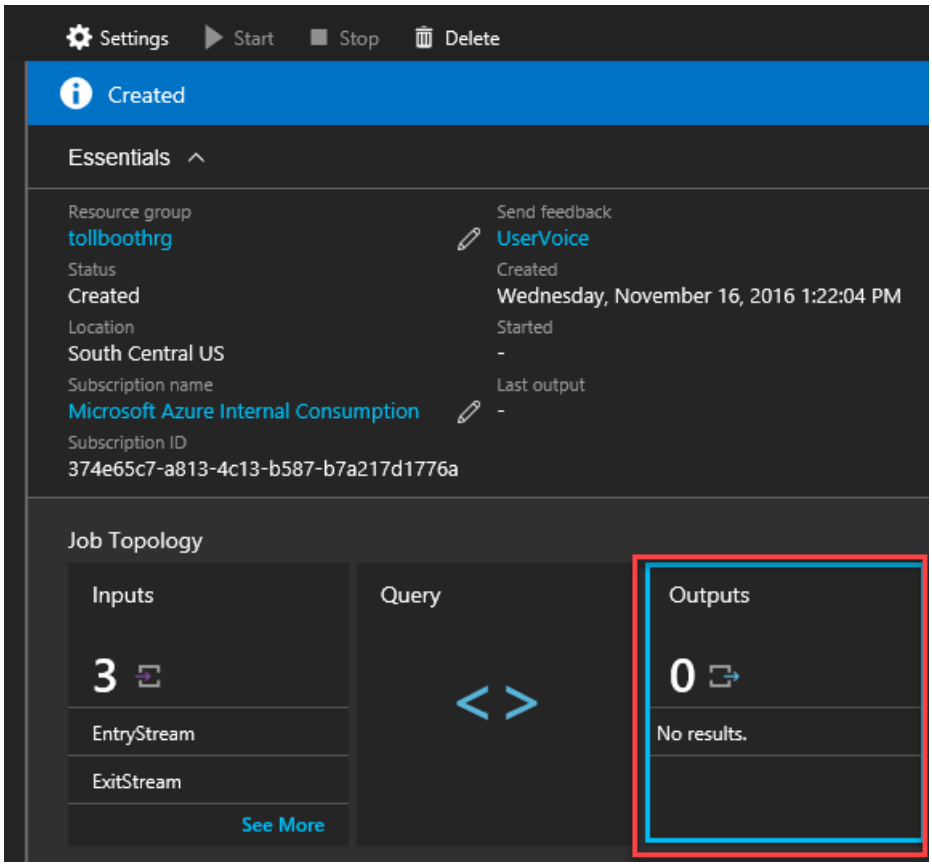


14. Click **Create** to finish the wizard.

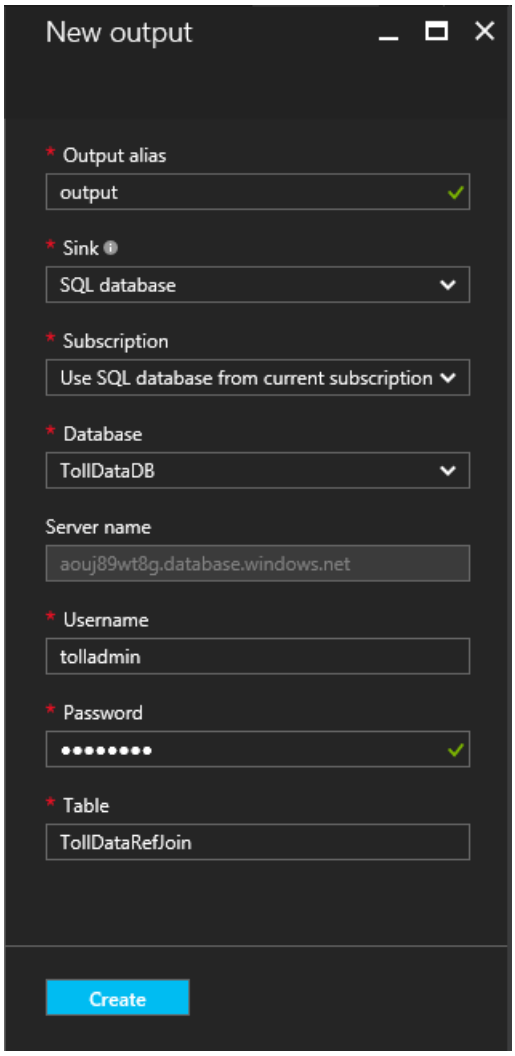
Now all inputs are defined.

Define output

1. On the Stream Analytics job overview pane, select **OUTPUTS**.



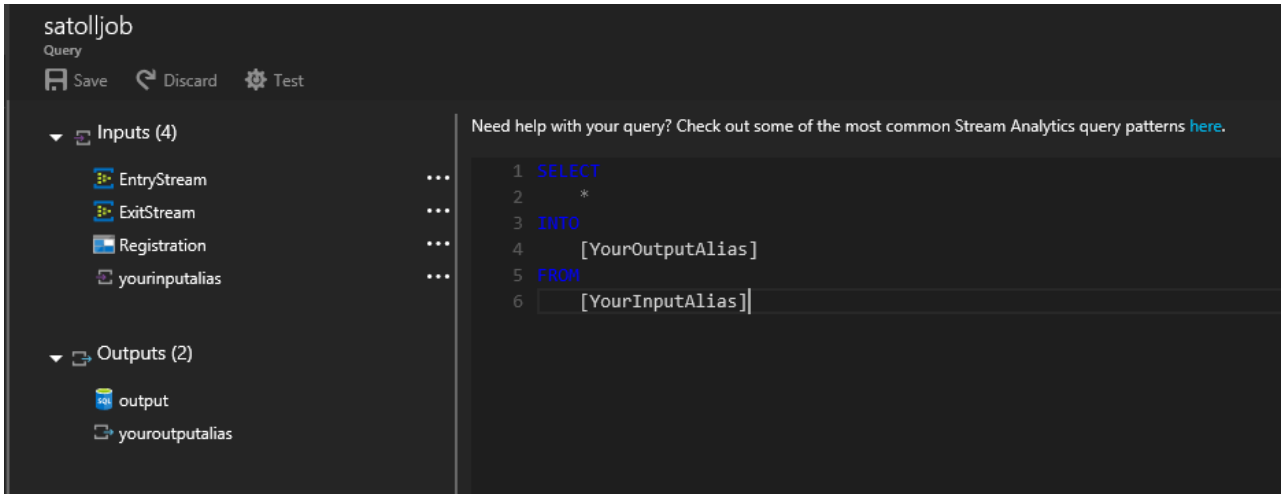
2. Click **Add**.
3. Set the **Output alias** to 'output' and then **Sink** to **SQL database**.
4. Select the server name that was used in the "Connect to Database from Visual Studio" section of the article. The database name is **TollDataDB**.
5. Enter **tolladmin** in the **USERNAME** field, **123toll!** in the **PASSWORD** field, and **TollDataRefJoin** in the **TABLE** field.



6. Click **Create**.

Azure Stream analytics query

The **QUERY** tab contains a SQL query that transforms the incoming data.



This tutorial attempts to answer several business questions that are related to toll data and constructs Stream Analytics queries that can be used in Azure Stream Analytics to provide a relevant answer.

Before you start your first Stream Analytics job, let's explore a few scenarios and the query syntax.

Introduction to Azure Stream Analytics query language

Let's say that you need to count the number of vehicles that enter a toll booth. Because this is a continuous stream of events, you have to define a "period of time." Let's modify the question to be "How many vehicles enter a toll booth every three minutes?". This is commonly referred to as the tumbling count.

Let's look at the Azure Stream Analytics query that answers this question:

```
SELECT TollId, System.Timestamp AS WindowEnd, COUNT(*) AS Count
FROM EntryStream TIMESTAMP BY EntryTime
GROUP BY TUMBLINGWINDOW(minute, 3), TollId
```

As you can see, Azure Stream Analytics uses a query language that's like SQL and adds a few extensions to specify time-related aspects of the query.

For more details, read about Time Management and Windowing constructs used in the query from MSDN.

Testing Azure Stream Analytics queries

Now that you have written your first Azure Stream Analytics query, it is time to test it by using sample data files located in your TollApp folder in the following path:

..\TollApp\TollApp\Data

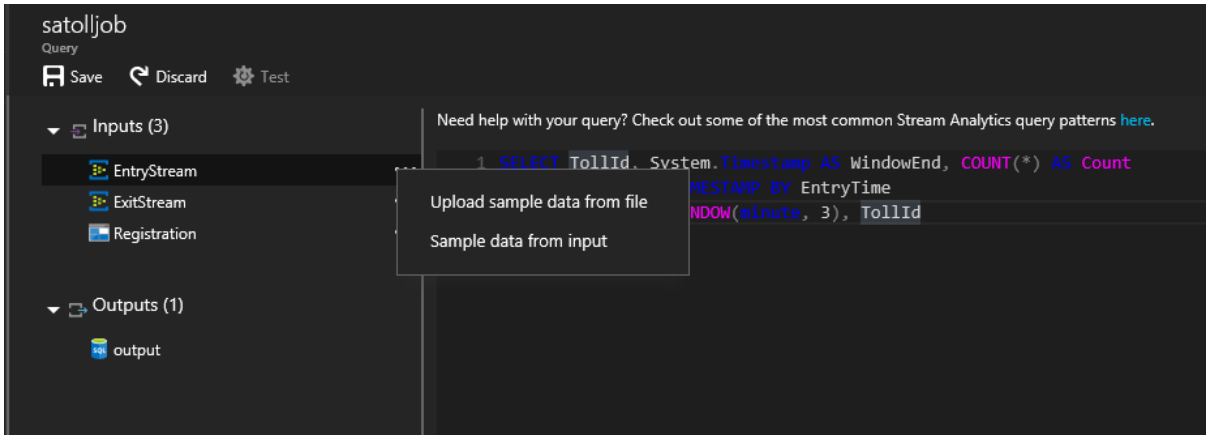
This folder contains the following files:

- Entry.json
- Exit.json
- Registration.json

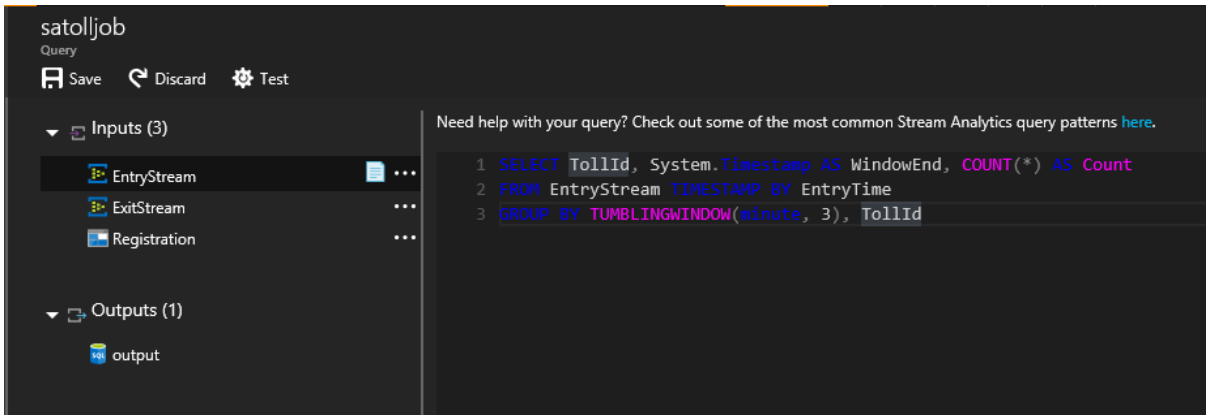
Question 1: Number of vehicles entering a toll booth

1. Open the Azure portal and go to your created Azure Stream Analytics job. Click the **QUERY** tab and paste query from the previous section.

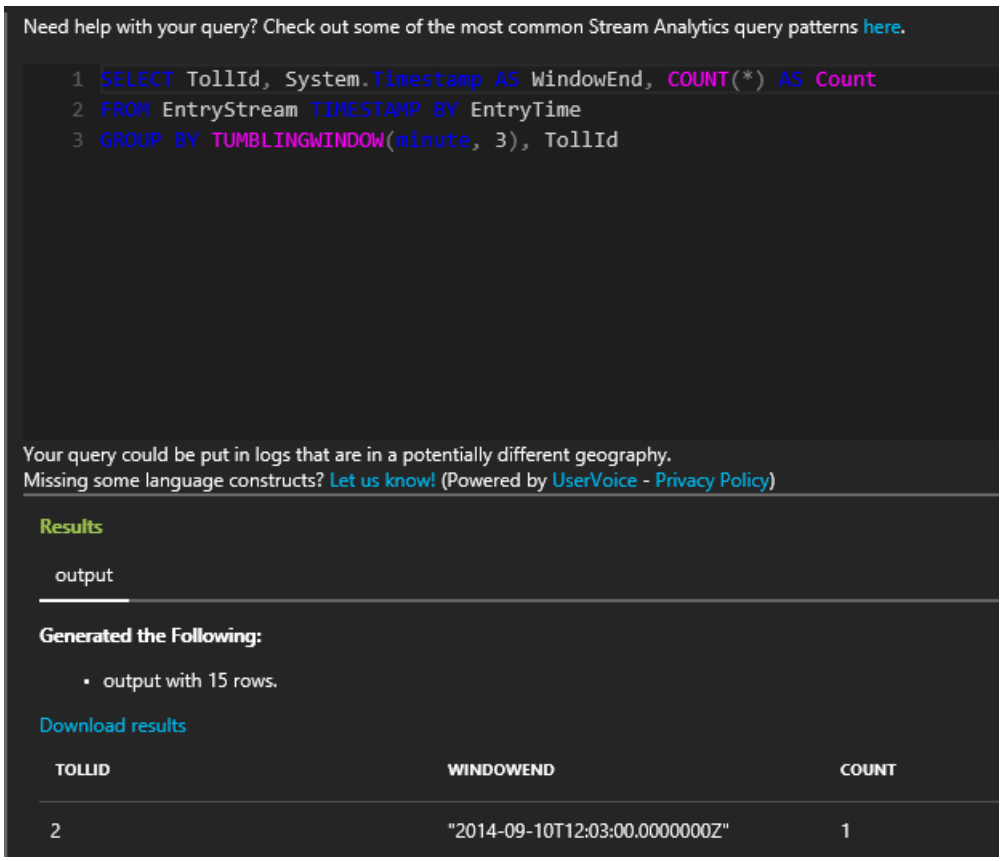
2. To validate this query against sample data, upload the data into the EntryStream input by clicking the ... symbol and selecting **Upload sample data from file**.



3. In the pane that appears select the file (Entry.json) on your local machine and click **OK**. The **Test** icon will now illuminate and be clickable.



4. Validate that the output of the query is as expected:



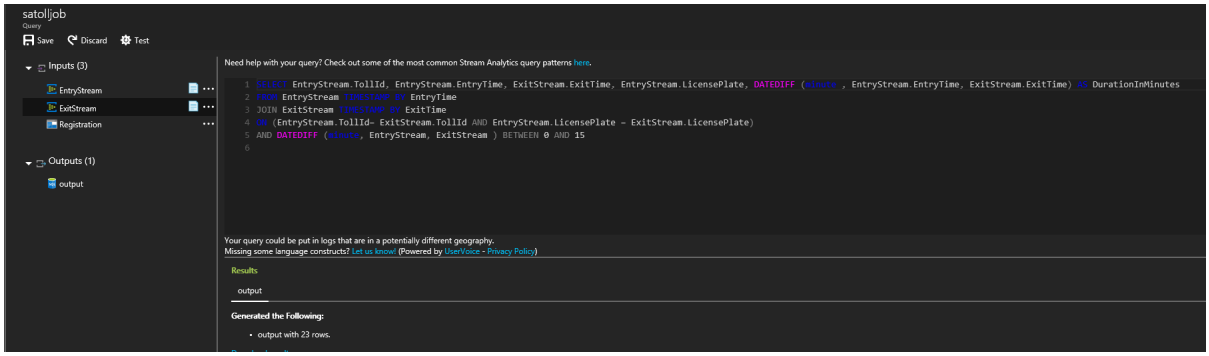
Question 2: Report total time for each car to pass through the toll booth

The average time that's required for a car to pass through the toll helps to assess the efficiency of the process and the customer experience.

To find the total time, you need to join the EntryTime stream with the ExitTime stream. You will join the streams on TollId and LicencePlate columns. The **JOIN** operator requires you to specify temporal leeway that describes the acceptable time difference between the joined events. You will use **DATEDIFF** function to specify that events should be no more than 15 minutes from each other. You will also apply the **DATEDIFF** function to exit and entry times to compute the actual time that a car spends in the toll station. Note the difference of the use of **DATEDIFF** when it's used in a **SELECT** statement rather than a **JOIN** condition.

```
SELECT EntryStream.TollId, EntryStream.EntryTime, ExitStream.ExitTime, EntryStream.LicensePlate, DATEDIFF
(minute , EntryStream.EntryTime, ExitStream.ExitTime) AS DurationInMinutes
FROM EntryStream TIMESTAMP BY EntryTime
JOIN ExitStream TIMESTAMP BY ExitTime
ON (EntryStream.TollId= ExitStream.TollId AND EntryStream.LicensePlate = ExitStream.LicensePlate)
AND DATEDIFF (minute, EntryStream, ExitStream ) BETWEEN 0 AND 15
```

1. To test this query, update the query on the **QUERY** for the job. Add the test file for **ExitStream** just like **EntryStream** was entered above.
2. Click **Test**.
3. Select the check box to test the query and view the output:



Question 3: Report all commercial vehicles with expired registration

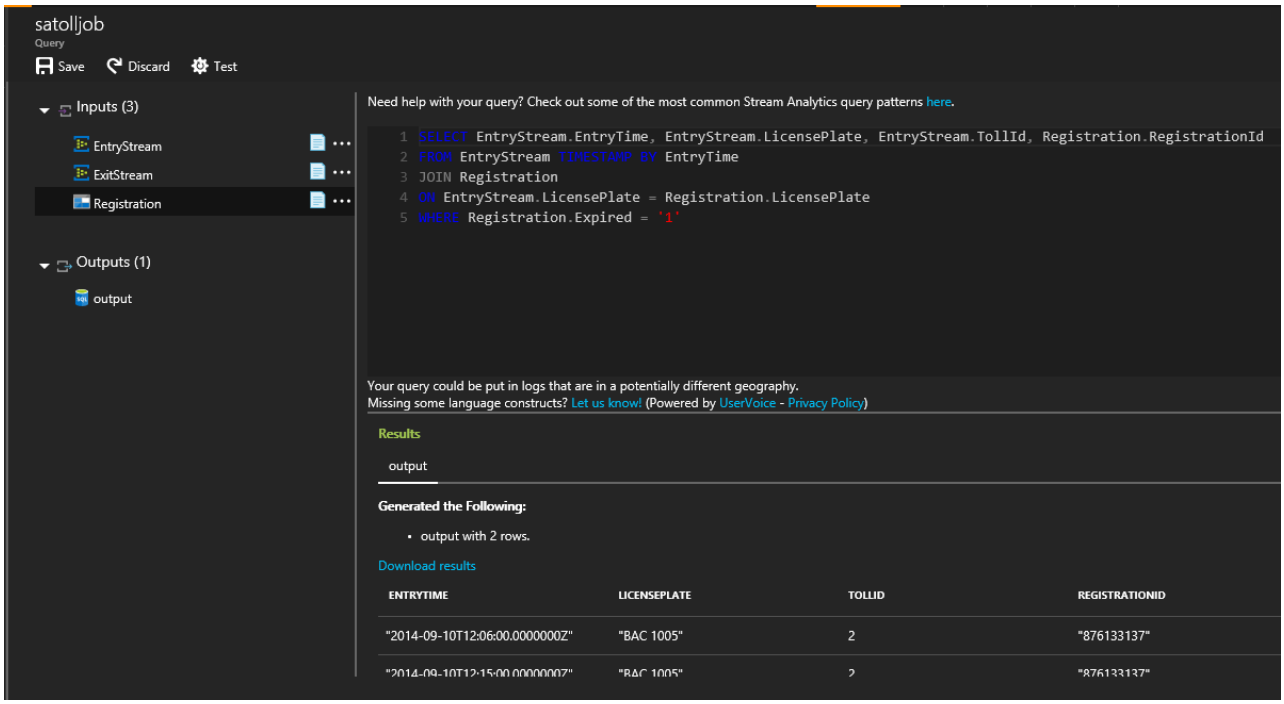
Azure Stream Analytics can use static snapshots of data to join with temporal data streams. To demonstrate this capability, use the following sample question.

If a commercial vehicle is registered with the toll company, it can pass through the toll booth without being stopped for inspection. You will use Commercial Vehicle Registration lookup table to identify all commercial vehicles that have expired registrations.

```
SELECT EntryStream.EntryTime, EntryStream.LicensePlate, EntryStream.TollId, Registration.RegistrationId
FROM EntryStream TIMESTAMP BY EntryTime
JOIN Registration
ON EntryStream.LicensePlate = Registration.LicensePlate
WHERE Registration.Expired = '1'
```

To test a query by using reference data, you need to define an input source for the reference data, which you have done already.

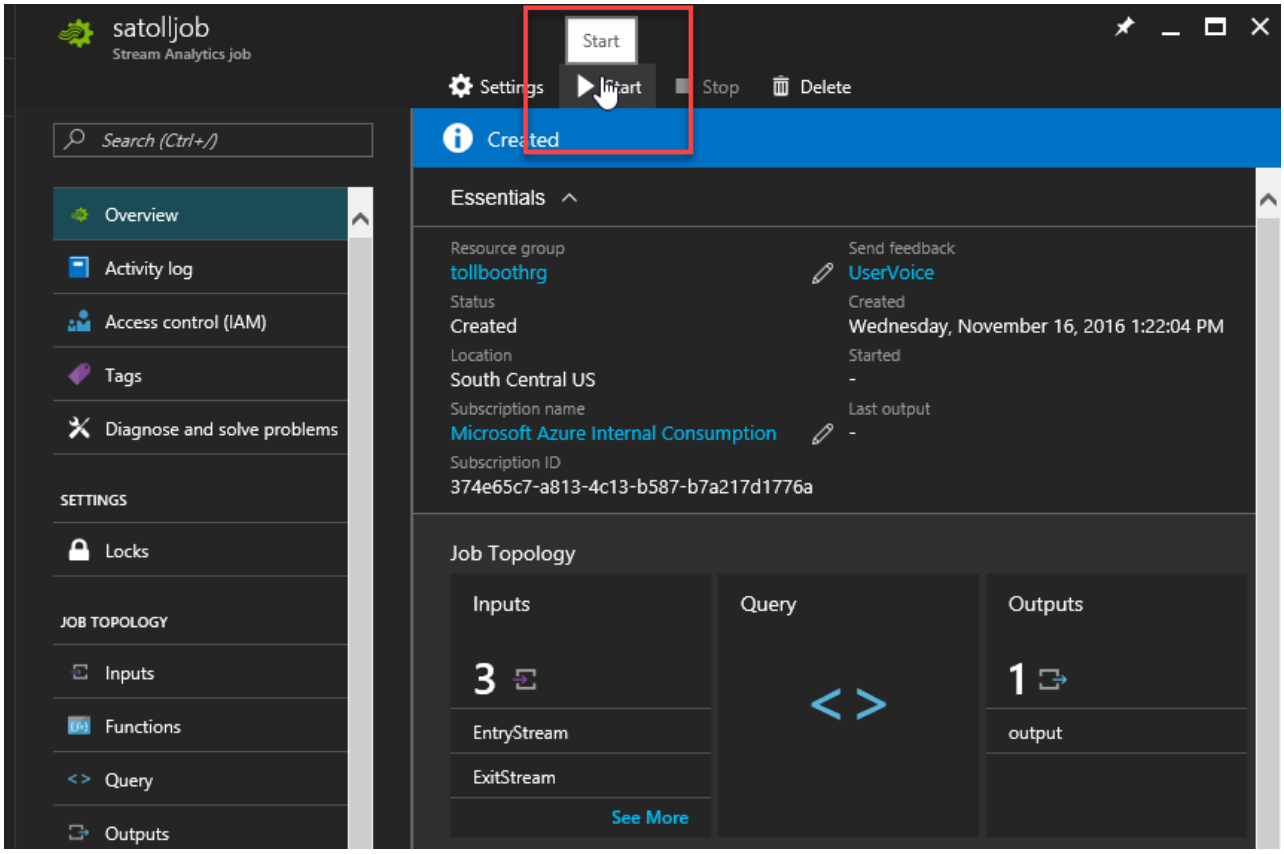
To test this query, paste the query into the **QUERY** tab, click **Test**, and specify the two input sources and the registration sample data and click **Test**.



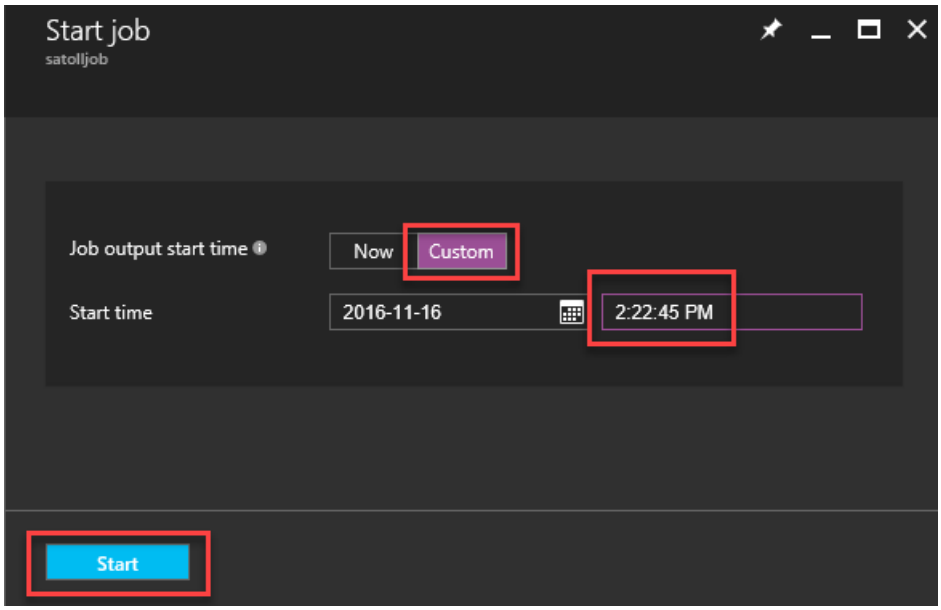
Start the Stream Analytics job

Now it's time to finish the configuration and start the job. Save the query from Question 3, which will produce output that matches the schema of the **TollDataRefJoin** output table.

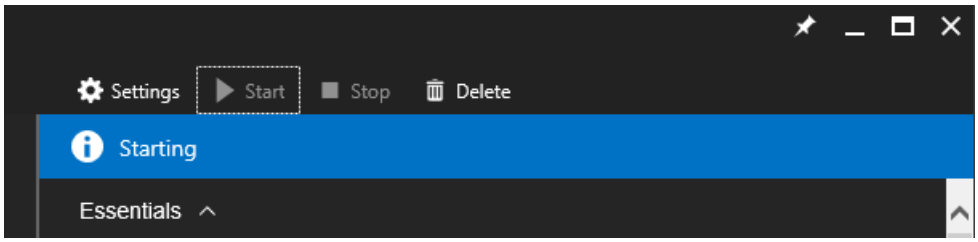
Go to the job **DASHBOARD**, and click **START**.



In the dialog box that opens, change the **START OUTPUT** time to **CUSTOM TIME**. Change the hour to one hour before the current time. This change ensures that all events from the event hub are processed since you started to generate the events at the beginning of the tutorial. Now click the **Start** button to start the job.

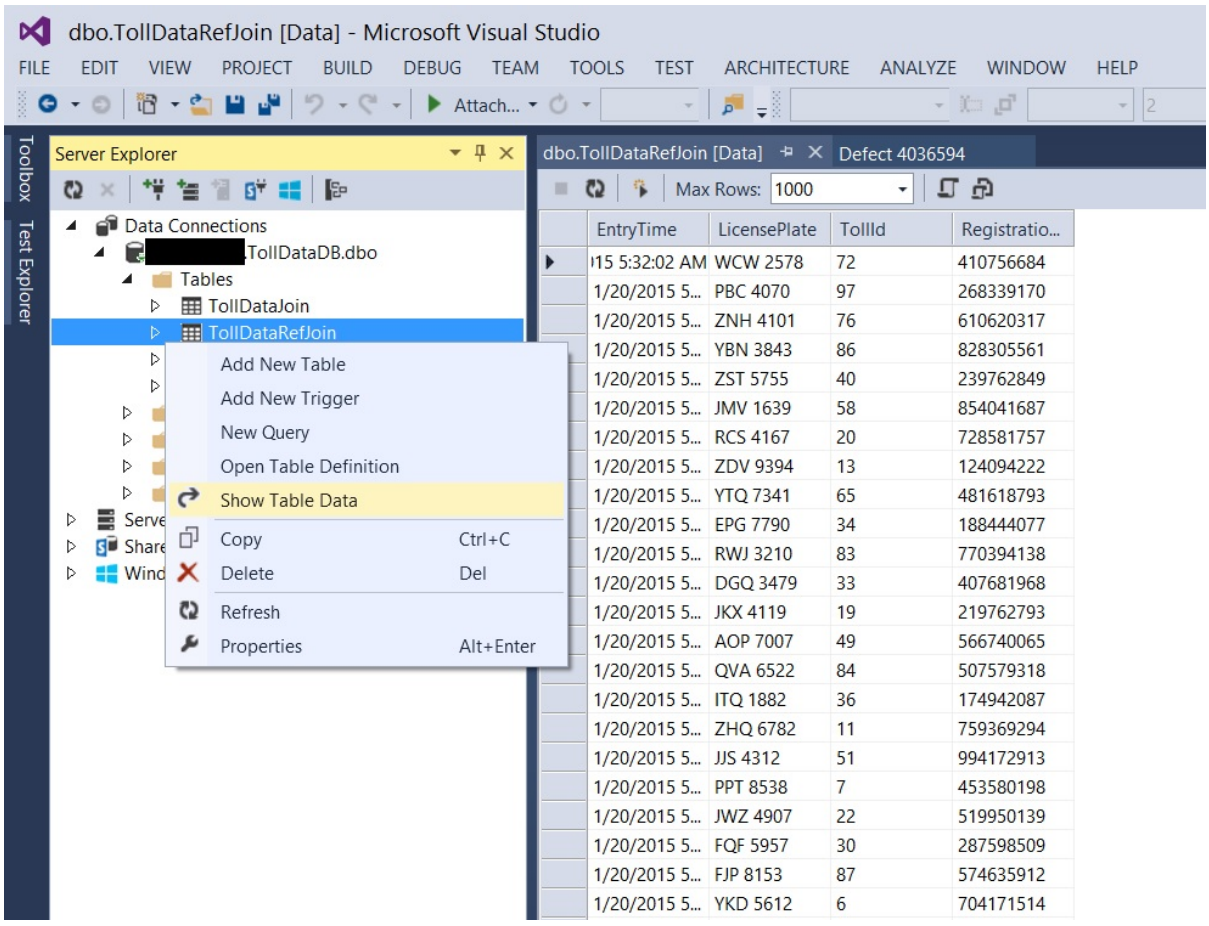


Starting the job can take a few minutes. You can see the status on the top-level page for Stream Analytics.



Check results in Visual Studio

1. Open Visual Studio Server Explorer, and right-click the **TollDataRefJoin** table.
2. Click **Show Table Data** to see the output of your job.



Scale out Azure Stream Analytics jobs

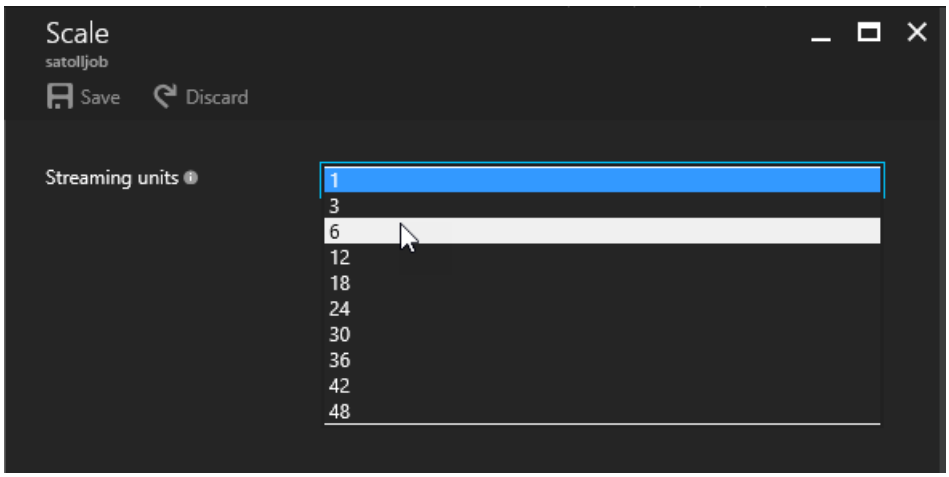
Azure Stream Analytics is designed to elastically scale so that it can handle a lot of data. The Azure Stream Analytics query can use a **PARTITION BY** clause to tell the system that this step will scale out. **PartitionId** is a special column that the system adds to match the partition ID of the input (event hub).

```
SELECT TollId, System.Timestamp AS WindowEnd, COUNT(*)AS Count
FROM EntryStream TIMESTAMP BY EntryTime PARTITION BY PartitionId
GROUP BY TUMBLINGWINDOW(minute,3), TollId, PartitionId
```

1. Stop the current job, update the query in the **QUERY** tab, and open the **Settings** gear in the job dashboard. Click **Scale**.

   STREAMING UNITS define the amount of compute power that the job can receive.

2. Change the drop down from 1 from 6.

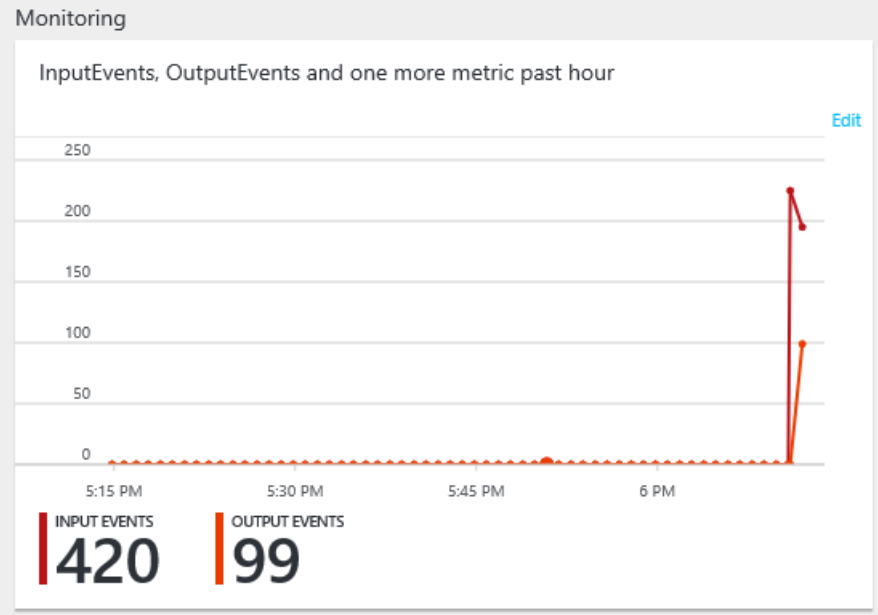


3. Go to the **OUTPUTS** tab and change the name of the SQL table to **TollDataTumblingCountPartitioned**.

If you start the job now, Azure Stream Analytics can distribute work across more compute resources and achieve better throughput. Please note that the TollApp application is also sending events partitioned by TollId.

Monitor

The **MONITOR** area contains statistics about the running job. First time configuration is needed to use the storage account in the same region (name toll like the rest of this document).



You can access **Activity Logs** from the job dashboard **Settings** area as well.

Conclusion

This tutorial introduced you to the Azure Stream Analytics service. It demonstrated how to configure inputs and outputs for the Stream Analytics job. Using the Toll Data scenario, the tutorial explained common types of problems that arise in the space of data in motion and how they can be solved with simple SQL-like queries in Azure Stream Analytics. The tutorial described SQL extension constructs for working with temporal data. It showed how to join data streams, how to enrich the data stream with static reference data, and how to scale out a query to achieve higher throughput.

Although this tutorial provides a good introduction, it is not complete by any means. You can find more query patterns using the SAQL language at Query examples for common Stream Analytics usage patterns. Refer to the online documentation to learn more about Azure Stream Analytics.

Clean up your Azure account

1. Stop the Stream Analytics job in the Azure portal.

   The Setup.ps1 script creates two event hubs and a SQL database. The following instructions help you clean up resources at the end of the tutorial.

2. In a PowerShell window, type **.\Cleanup.ps1** to start the script that deletes resources used in the tutorial.

> **NOTE**
> Resources are identified by the name. Make sure you carefully review each item before confirming removal.

Social media analysis: Real-time Twitter sentiment analysis in Azure Stream Analytics

2/3/2017 • 11 min to read • Edit on GitHub

Learn how to build a sentiment analysis solution for social media analytics by bringing real-time Twitter events into Azure Event Hubs. You'll write an Azure Stream Analytics query to analyze the data. You'll then either store the results for later perusal or use a dashboard and Power BI to provide insights in real time.

Social media analytics tools help organizations understand trending topics, that is, subjects and attitudes that have a high volume of posts in social media. Sentiment analysis, which is also called *opinion mining*, uses social media analytics tools to determine attitudes toward a product, idea, and so on. Real-time Twitter trend analysis is a great example because the hashtag subscription model enables you to listen to specific keywords and develop sentiment analysis on the feed.

Scenario: Sentiment analysis in real time

A company that has a news media website is interested in getting an advantage over its competitors by featuring site content that is immediately relevant to its readers. The company uses social media analysis on topics that are relevant to readers by doing real-time sentiment analysis on Twitter data. Specifically, to identify trending topics in real time on Twitter, the company needs real-time analytics about the tweet volume and sentiment for key topics. So, in essence, the need is a sentiment analysis analytics engine that's based on this social media feed.

Prerequisites

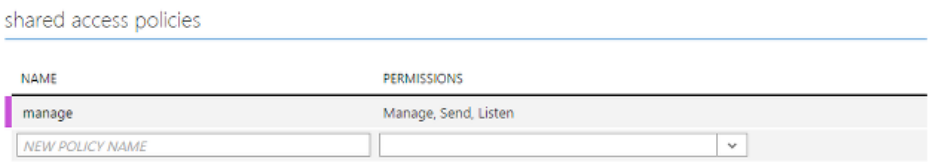
- Twitter account and OAuth access token
- TwitterClient.zip from the Microsoft Download Center
- Optional: Source code for twitter client from GitHub

Create an event hub input and a consumer group

The sample application will generate events and push them to an Event Hubs instance (an event hub, for short). Service Bus event hubs are the preferred method of event ingestion for Stream Analytics. See Event Hubs documentation in Azure Service Bus documentation.

Use the following steps to create an event hub.

1. In the Azure portal, click **NEW** > **APP SERVICES** > **SERVICE BUS** > **EVENT HUB** > **QUICK CREATE**, and provide a name, region, and new or existing namespace.
2. As a best practice, each Stream Analytics job should read from a single Event Hubs consumer group. We will walk you through the process of creating a consumer group later. You can learn more about consumer groups at Azure Event Hubs overview. To create a consumer group, go to the newly created event hub, click the **CONSUMER GROUPS** tab, click **CREATE** on the bottom of the page, and then provide a name for your consumer group.
3. To grant access to the event hub, we will need to create a shared access policy. Click the **CONFIGURE** tab of your event hub.
4. Under **SHARED ACCESS POLICIES**, create a new policy with **MANAGE** permissions.



5. Click **SAVE** at the bottom of the page.
6. Go to the **DASHBOARD**, click **CONNECTION INFORMATION** at the bottom of the page, and then copy and save the

connection information. (Use the copy icon that appears under the search icon.)

Configure and start the Twitter client application

We have provided a client application that will connect to Twitter data via Twitter's Streaming APIs to collect Tweet events about a parameterized set of topics. The Sentiment140 open source tool is used to assign a sentiment value to each tweet.

- 0 = negative
- 2 = neutral
- 4 = positive

Then, Tweet events are pushed to the event hub.

Follow these steps to set up the application:

1. Download the TwitterClient solution.
2. Open TwitterClient.exe.config, and replace oauth_consumer_key, oauth_consumer_secret, oauth_token, and oauth_token_secret with Twitter tokens that have your values.

   Steps to generate an OAuth access token

   Note that you will need to make an empty application to generate a token.

3. Replace the EventHubConnectionString and EventHubName values in TwitterClient.exe.config with the connection string and name of your event hub. The connection string that you copied earlier gives you both the connection string and the name of your event hub, so be sure to separate them and put each in the correct field. For example, consider the following connection string:

   ```
   Endpoint=sb://your.servicebus.windows.net/;SharedAccessKeyName=yourpolicy;SharedAccessKey=yoursharedaccesskey;EntityPath=yourhub
   ```

   The TwitterClient.exe.config file should contain your settings as in the following example:

   ```
     add key="EventHubConnectionString"
   value="Endpoint=sb://your.servicebus.windows.net/;SharedAccessKeyName=yourpolicy;SharedAccessKey=yoursharedaccess
   key"
     add key="EventHubName" value="yourhub"
   ```

   It is important to note that the text "EntityPath=" does **not** appear in the EventHubName value.

4. *Optional:* Adjust the keywords to search for. As a default, this application looks for "Azure,Skype,XBox,Microsoft,Seattle". You can adjust the values for **twitter_keywords** in TwitterClient.exe.config, if desired.

5. Run TwitterClient.exe to start your application. You will see Tweet events with the **CreatedAt**, **Topic**, and **SentimentScore** values being sent to your event hub.

Create a Stream Analytics job

Now that Tweet events are streaming in real time from Twitter, we can set up a Stream Analytics job to analyze these events in real time.

Provision a Stream Analytics job

1. In the Azure portal, click **NEW** > **DATA SERVICES** > **STREAM ANALYTICS** > **QUICK CREATE**.
2. Specify the following values, and then click **CREATE STREAM ANALYTICS JOB**:

   - **JOB NAME**: Enter a job name.
   - **REGION**: Select the region where you want to run the job. Consider placing the job and the event hub in the same

region to ensure better performance and to ensure that you will not be paying to transfer data between regions.

- **STORAGE ACCOUNT**: Choose the Azure storage account that you would like to use to store monitoring data for all Stream Analytics jobs that run within this region. You have the option to choose an existing storage account or to create a new one.

3. Click **STREAM ANALYTICS** in the left pane to list the Stream Analytics jobs.

The new job will be shown with a status of **CREATED**. Notice that the **START** button on the bottom of the page is disabled. You must configure the job input, output, and query before you can start the job.

Specify job input

1. In your Stream Analytics job, click **INPUTS** from the top of the page, and then click **ADD INPUT**. The dialog box that opens will walk you through a number of steps to set up your input.
2. Click **DATA STREAM**, and then click the right button.
3. Click **EVENT HUB**, and then click the right button.
4. Type or select the following values on the third page:

   - **INPUT ALIAS**: Enter a friendly name for this job input, such as *TwitterStream*. Note that you will use this name in the query later.
   - **EVENT HUB**: If the event hub that you created is in the same subscription as the Stream Analytics job, select the namespace that the event hub is in.
     - If your event hub is in a different subscription, click **Use Event Hub from Another Subscription**, and then manually enter information for **SERVICE BUS NAMESPACE**, **EVENT HUB NAME**, **EVENT HUB POLICY NAME**, **EVENT HUB POLICY KEY**, and **EVENT HUB PARTITION COUNT**.
   - **EVENT HUB NAME**: Select the name of the event hub.
   - **EVENT HUB POLICY NAME**: Select the event hub policy that you created earlier in this tutorial.
   - **EVENT HUB CONSUMER GROUP**: Type the name of the consumer group that you created earlier in this tutorial.

5. Click the right button.
6. Specify the following values:

   - **EVENT SERIALIZER FORMAT**: JSON
   - **ENCODING**: UTF8

7. Click the **CHECK** button to add this source and to verify that Stream Analytics can successfully connect to the event hub.

Specify job query

Stream Analytics supports a simple, declarative query model that describes transformations. To learn more about the language, see the Azure Stream Analytics Query Language Reference. This tutorial will help you author and test several queries over Twitter data.

Sample data input

To validate your query against actual job data, you can use the **SAMPLE DATA** feature to extract events from your stream and create a .json file of the events for testing.

1. Select your event hub input, and then click **SAMPLE DATA** at the bottom of the page.
2. In the dialog box that opens, specify a **START TIME** to start collecting data and a **DURATION** for how much additional data to consume.
3. Click the **DETAILS** button, and then click the **Click here** link to download and save the generated .json file.

Pass-through query

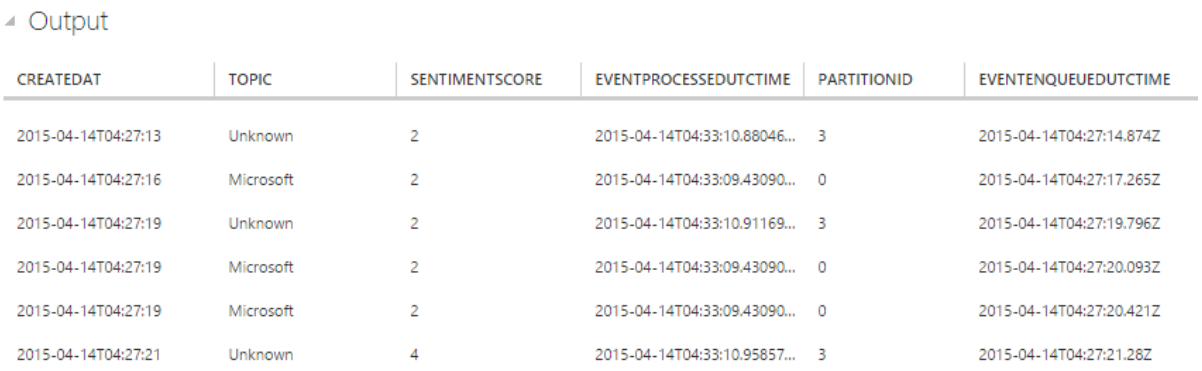
To start, we will do a simple pass-through query that projects all the fields in an event.

1. Click **QUERY** at the top of the Stream Analytics job page.
2. In the code editor, replace the initial query template with the following:

```
SELECT * FROM TwitterStream
```

Make sure that the name of the input source matches the name of the input that you specified earlier.

3. Click **Test** under the query editor.
4. Go to your sample .json file.
5. Click the **CHECK** button, and see the results below the query definition.



Output

CREATEDAT	TOPIC	SENTIMENTSCORE	EVENTPROCESSEDUTCTIME	PARTITIONID	EVENTENQUEUEDUTCTIME
2015-04-14T04:27:13	Unknown	2	2015-04-14T04:33:10.88046...	3	2015-04-14T04:27:14.874Z
2015-04-14T04:27:16	Microsoft	2	2015-04-14T04:33:09.43090...	0	2015-04-14T04:27:17.265Z
2015-04-14T04:27:19	Unknown	2	2015-04-14T04:33:10.91169...	3	2015-04-14T04:27:19.796Z
2015-04-14T04:27:19	Microsoft	2	2015-04-14T04:33:09.43090...	0	2015-04-14T04:27:20.093Z
2015-04-14T04:27:19	Microsoft	2	2015-04-14T04:33:09.43090...	0	2015-04-14T04:27:20.421Z
2015-04-14T04:27:21	Unknown	4	2015-04-14T04:33:10.95857...	3	2015-04-14T04:27:21.28Z

Count of tweets by topic: Tumbling window with aggregation

To compare the number of mentions among topics, we'll use a TumblingWindow to get the count of mentions by topic every five seconds.

1. Change the query in the code editor to:

```
SELECT System.Timestamp as Time, Topic, COUNT(*)
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY TUMBLINGWINDOW(s, 5), Topic
```

   This query uses the **TIMESTAMP BY** keyword to specify a timestamp field in the payload to be used in the temporal computation. If this field wasn't specified, the windowing operation would be performed by using the time that each event arrived at the event hub. Learn more in the "Arrival Time Vs Application Time" section of Stream Analytics Query Reference.

   This query also accesses a timestamp for the end of each window by using the **System.Timestamp** property.

2. Click **Rerun** under the query editor to see the results of the query.

Identify trending topics: Sliding window

To identify trending topics, we'll look for topics that cross a threshold value for mentions in a given amount of time. For the purposes of this tutorial, we'll check for topics that are mentioned more than 20 times in the last five seconds by using a SlidingWindow.

1. Change the query in the code editor to:

```
SELECT System.Timestamp as Time, Topic, COUNT(*) as Mentions
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY SLIDINGWINDOW(s, 5), topic
HAVING COUNT(*) > 20
```

2. Click **Rerun** under the query editor to see the results of the query.

Output

TIME	TOPIC	MENTIONS
2015-04-14T11:34:08.000Z	Microsoft	22
2015-04-14T12:00:37.000Z	Microsoft	24

Count of mentions and sentiment: Tumbling window with aggregation

The final query that we will test uses **TumblingWindow** to get the number of mentions, average, minimum, maximum, and

standard deviation of sentiment score for each topic every five seconds.

1. Change the query in the code editor to:

```
SELECT System.Timestamp as Time, Topic, COUNT(*), AVG(SentimentScore), MIN(SentimentScore),
Max(SentimentScore), STDEV(SentimentScore)
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY TUMBLINGWINDOW(s, 5), Topic
```

2. Click **Rerun** under the query editor to see the results of the query.
3. This is the query that we will use for our dashboard. Click **SAVE** at the bottom of the page.

Create output sink

Now that we have defined an event stream, an event hub input to ingest events, and a query to perform a transformation over the stream, the last step is to define an output sink for the job. We'll write the aggregated tweet events from our job query to Azure Blob storage. You could also push your results to Azure SQL Database, Azure Table storage, or Event Hubs, depending on your specific application needs.

Use the following steps to create a container for Blob storage, if you don't already have one:

1. Use an existing storage account or create a new storage account by clicking **NEW** > **DATA SERVICES** > **STORAGE** > **QUICK CREATE**, and then following the instructions on the screen.
2. Select the storage account, click **CONTAINERS** at the top of the page, and then click **ADD**.
3. Specify a **NAME** for your container, and set its **ACCESS** to **Public Blob**.

Specify job output

1. In your Stream Analytics job, click **OUTPUT** at the top of the page, and then click **ADD OUTPUT**. The dialog box that opens will walk you through several steps to set up your output.
2. Click **BLOB STORAGE**, and then click the right button.
3. Type or select the following values on the third page:
   - **OUTPUT ALIAS**: Enter a friendly name for this job output.
   - **SUBSCRIPTION**: If the Blob storage that you created is in the same subscription as the Stream Analytics job, click **Use Storage Account from Current Subscription**. If your storage is in a different subscription, click **Use Storage Account from Another Subscription**, and manually enter information for **STORAGE ACCOUNT**, **STORAGE ACCOUNT KEY**, and **CONTAINER**.
   - **STORAGE ACCOUNT**: Select the name of the storage account.
   - **CONTAINER**: Select the name of the container.
   - **FILENAME PREFIX**: Type a file prefix to use when writing blob output.
4. Click the right button.
5. Specify the following values:
   - **EVENT SERIALIZER FORMAT**: JSON
   - **ENCODING**: UTF8
6. Click the **CHECK** button to add this source and to verify that Stream Analytics can successfully connect to the storage account.

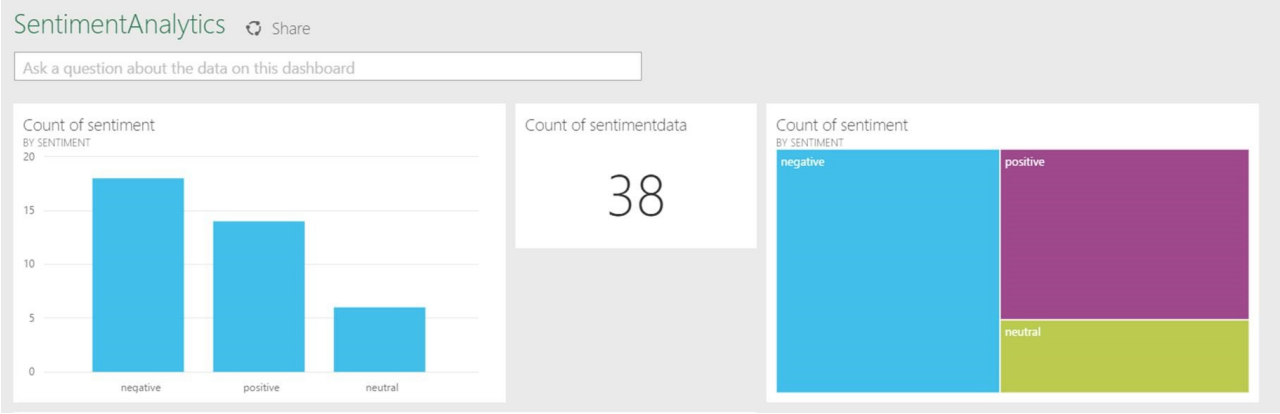
Start job

Because a job input, query, and output have all been specified, we are ready to start the Stream Analytics job.

1. From the job **DASHBOARD**, click **START** at the bottom of the page.
2. In the dialog box that opens, click **JOB START TIME**, and then click the **CHECK** button on the bottom of the dialog box. The job status will change to **Starting** and will shortly change to **Running**.

View output for sentiment analysis

After your job is running and processing the real-time Twitter stream, choose how you want to view the output for sentiment analysis. Use a tool like Azure Storage Explorer or Azure Explorer to view your job output in real time. From here, you can use Power BI to extend your application to include a customized dashboard like the one in the following screenshot.



Get support

For further assistance, try our Azure Stream Analytics forum.

Next steps

- Introduction to Azure Stream Analytics
- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Sentiment analysis by using Azure Stream Analytics and Azure Machine Learning

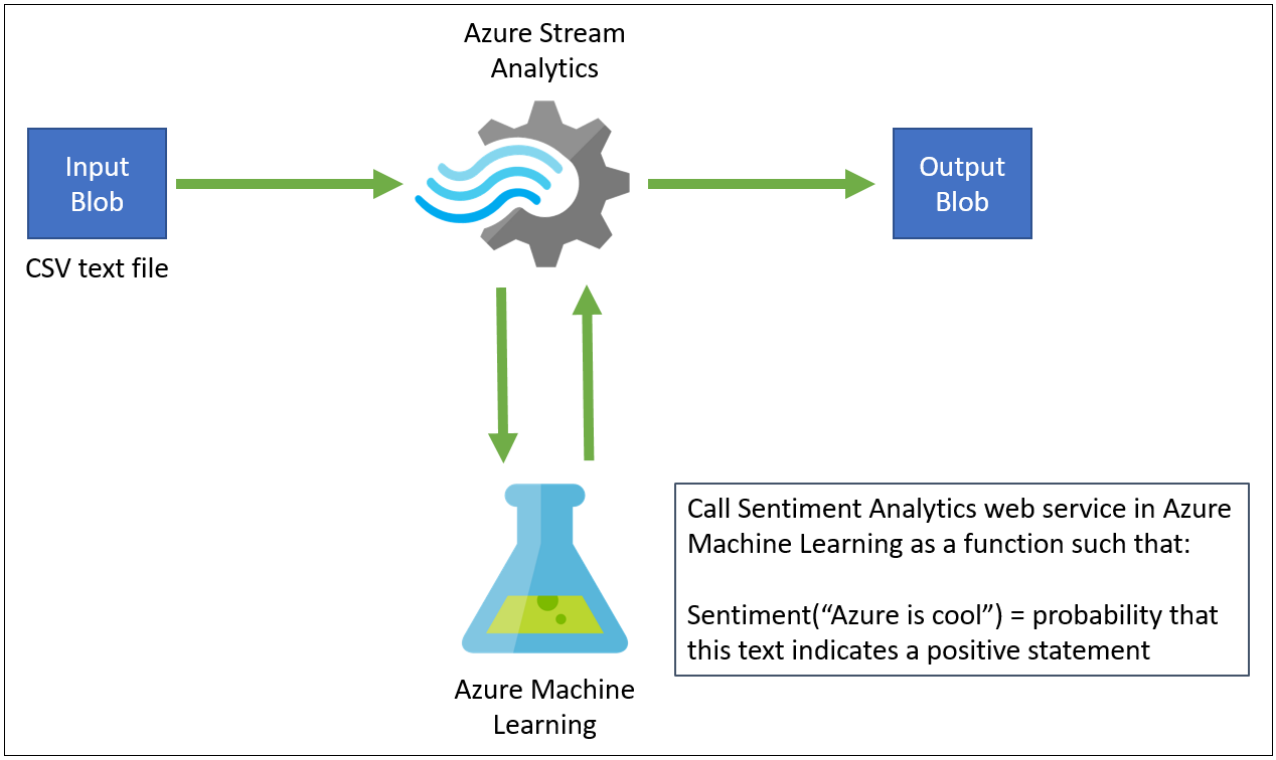
2/13/2017 • 6 min to read • Edit on GitHub

This article is designed to help you quickly set up a simple Azure Stream Analytics job, with Azure Machine Learning integration. We will use a sentiment analytics Machine Learning model from the Cortana Intelligence Gallery to analyze streaming text data, and determine the sentiment score in real time. The information in this article can help you understand scenarios such as real-time sentiment analytics on streaming Twitter data, analyze records of customer chats with support staff, and evaluate comments on forums, blogs, and videos, in addition to many other real-time, predictive scoring scenarios.

This article offers a sample CSV file with text as input in Azure Blob storage, shown in the following image. The job applies the sentiment analytics model as a user-defined function (UDF) on the sample text data from the blob store. The end result is placed in the same blob store in another CSV file.

Text
Azure is cool
Cortana Analytics rocks
I hate going to the gym

The following image demonstrates this configuration. For a more realistic scenario, you can replace Blob storage with streaming Twitter data from an Azure Event Hubs input. Additionally, you could build a Microsoft Power BI real-time visualization of the aggregate sentiment.



Prerequisites

The prerequisites for completing the tasks that are demonstrated in this article are as follows:

- An active Azure subscription.
- A CSV file with some data in it. You can download the file shown in Figure 1 from GitHub, or you can create your own file. For this article, we assume that you use the one provided for download on GitHub.

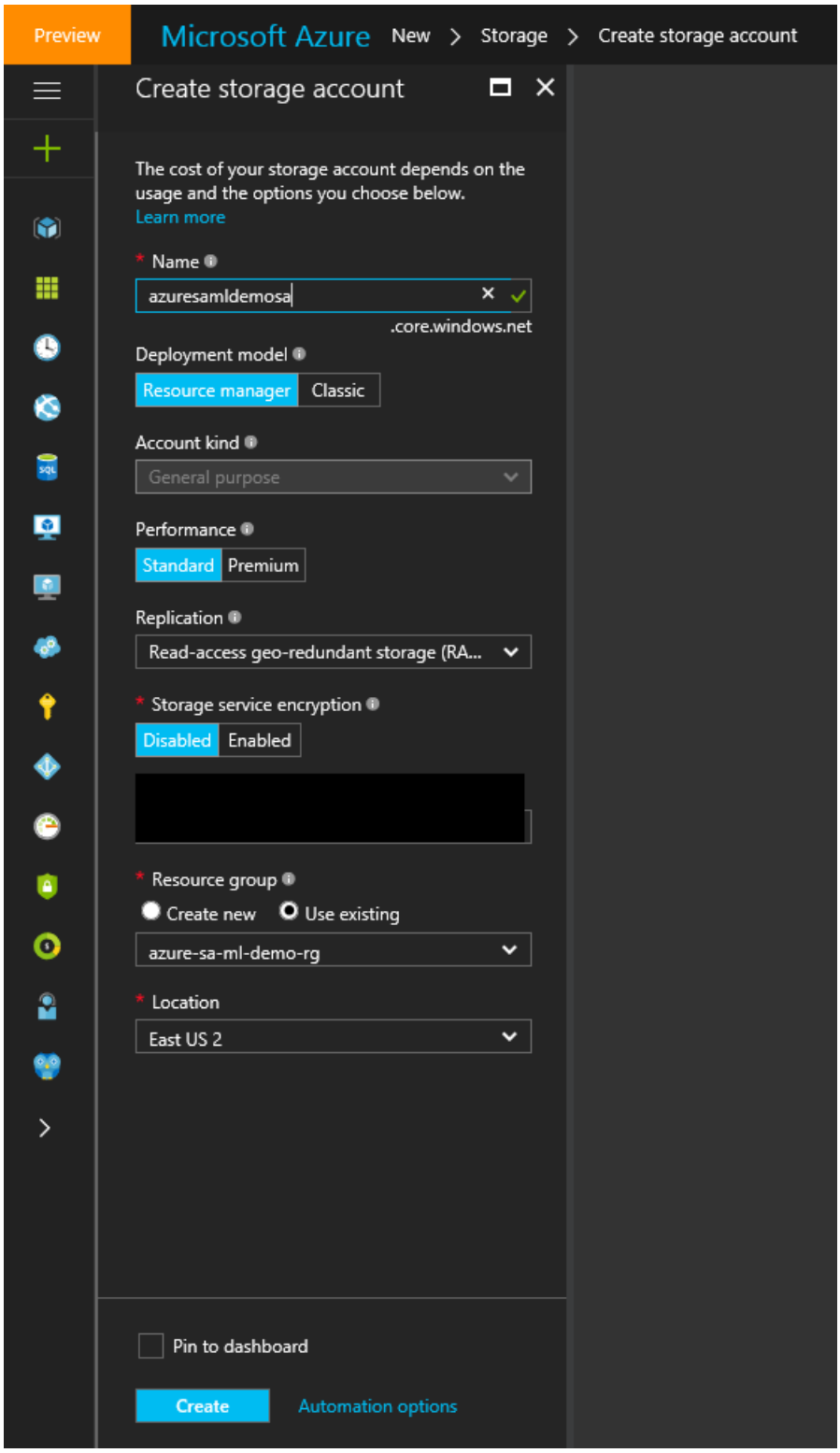
At a high level, to complete the tasks demonstrated in this article, you'll do the following:

1. Upload the CSV input file to Azure Blob storage.
2. Add a sentiment analytics model from the Cortana Intelligence Gallery to your Azure Machine Learning workspace.
3. Deploy this model as a web service in the Machine Learning workspace.
4. Create a Stream Analytics job that calls this web service as a function, to determine sentiment for the text input.
5. Start the Stream Analytics job and observe the output.

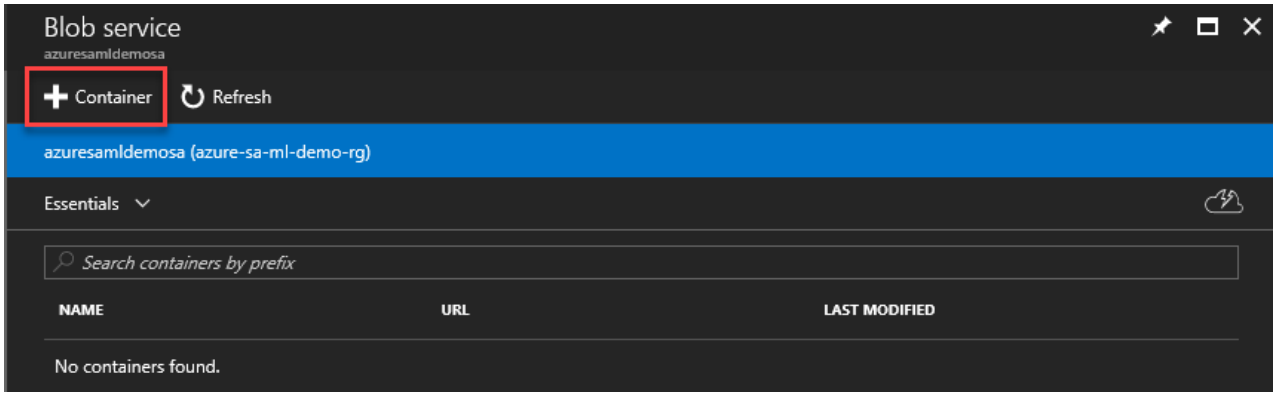
Create a storage blob and upload the CSV input file

For this step, you can use any CSV file, such as the one already specified as available for download on GitHub. Uploading the csv file is simple as it is an option included in creating a storage blob.

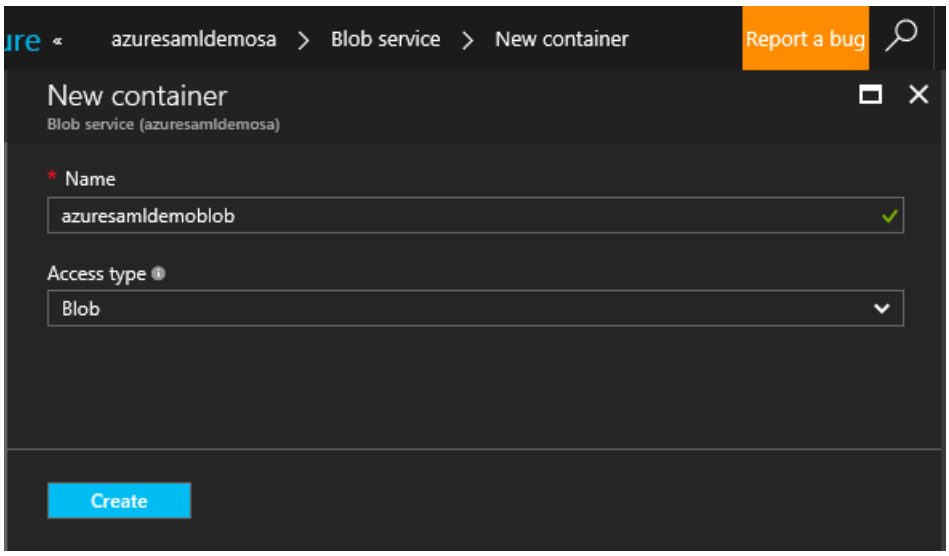
For our tutorial, create a new storage account by clicking **New** and then searching for 'storage account' and then selecting the resulting icon for storage account and providing details for the creation of the account. Provide a **Name** (azuresamldemosa in my example), create or use an existing **Resource group** and specify a **Location** (for location, it is important that all the resources created in this demo all use the same location if possible).



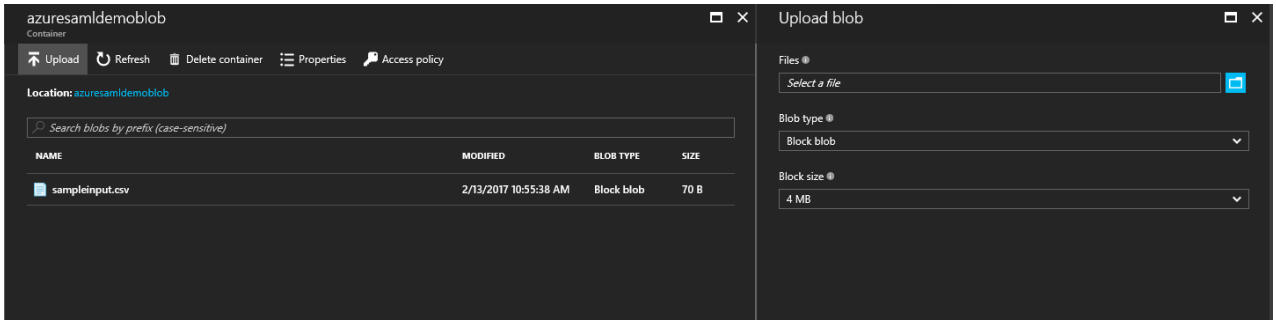
Once that is completed you can click on Blob service and create a blob container.



Then provide a **Name** for the container (azuresamldemoblob in my example) and verify the **Access type** is set to 'blob'.



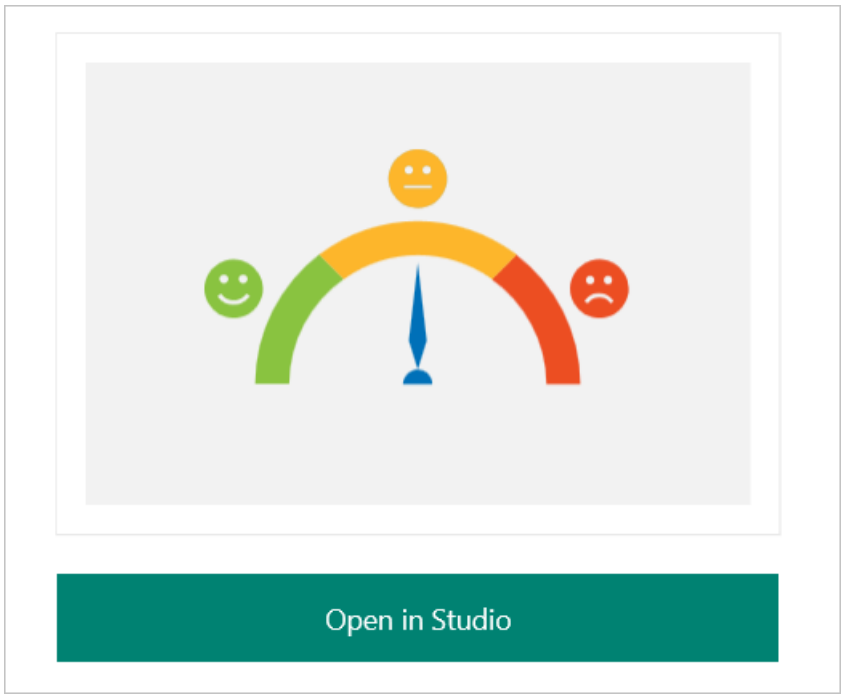
Now we can populate the blob with our data. Select **Files** and then select the file on your local drive that you downloaded from GitHub. I selected Block blob and 4 MB as a size these should be fine for this demonstration. Then select **Upload** and the portal will create a blob with the text sample for you.



Now that the sample data is in a blob it is time to enable the sentiment analysis model in Cortana Intelligence Gallery.

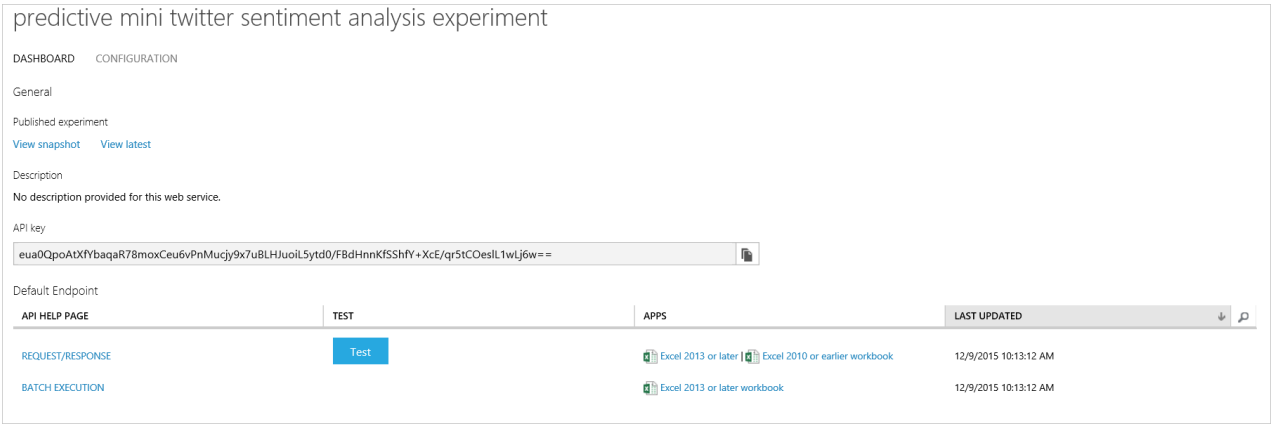
Add the sentiment analytics model from the Cortana Intelligence Gallery

1. Download the predictive sentiment analytics model from the Cortana Intelligence Gallery.
2. In Machine Learning Studio, select **Open in Studio**.



3. Sign in to go to the workspace. Select the location that best suits your own location.
4. Click **Run** at the bottom of the page.
5. After the process runs successfully, select **Deploy Web Service**.
6. The sentiment analytics model is ready to use. To validate, select the **Test** button and provide text input, such as, "I love Microsoft." The test should return a result similar to the following:

```
'Predictive Mini Twitter sentiment analysis Experiment' test returned ["4","0.715057671070099"]...
```

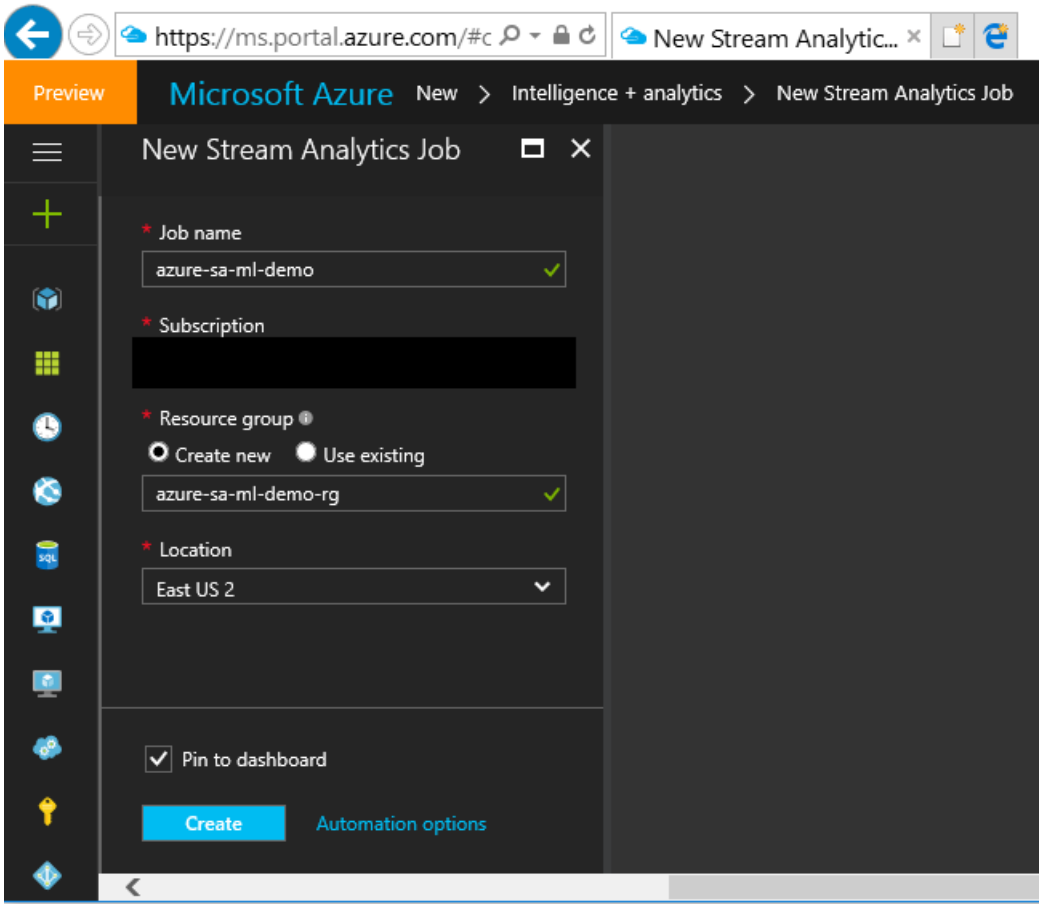


In the **Apps** column, select the link for **Excel 2010 or earlier workbook** to get your API key and the URL that you'll need later to set up the Stream Analytics job. (This step is required only to use a Machine Learning model from another Azure account workspace. This article assumes this is the case, to address that scenario.)

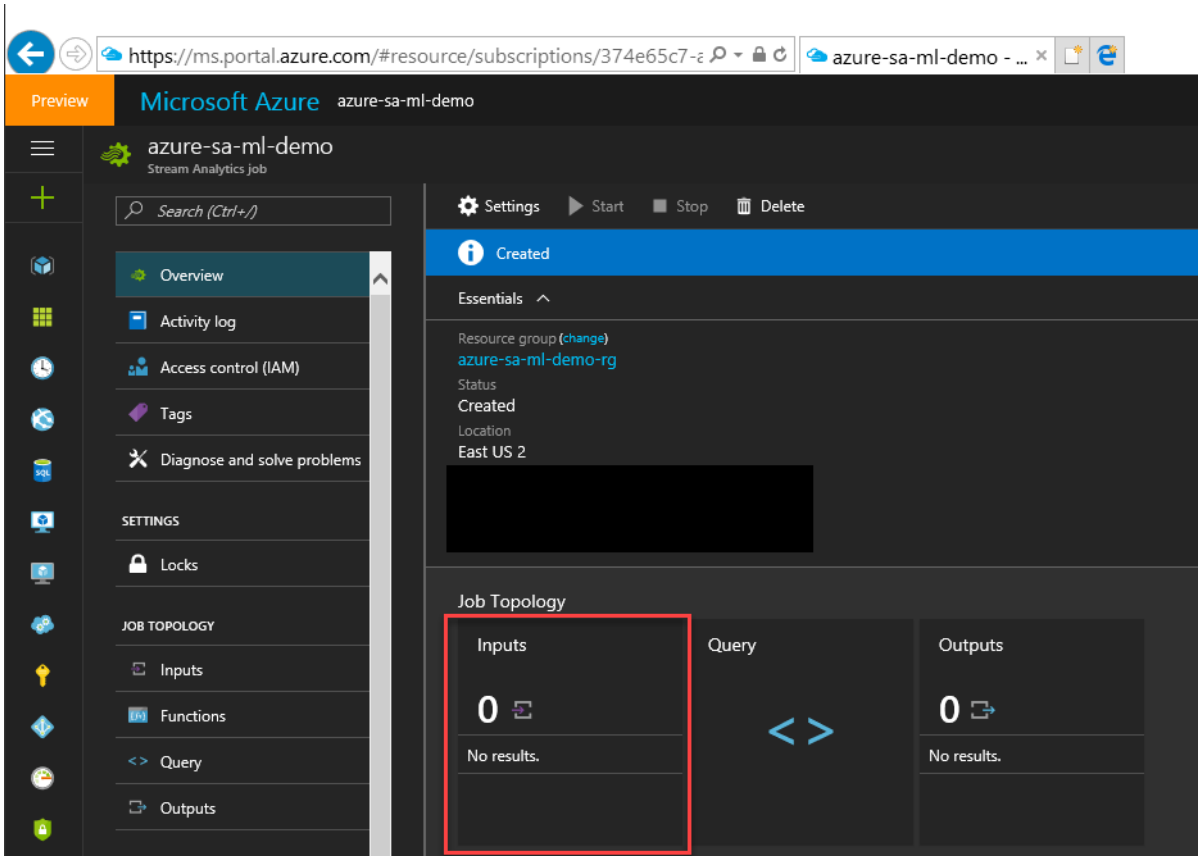
Note the web service URL and access key from the downloaded Excel file, as shown below:

Create a Stream Analytics job that uses the Machine Learning model

1. Go to the Azure portal.
2. Click **New** > **Intelligence + analytics** > **Stream Analytics**. Enter a name for your job in **Job name**, specify an existing resource group or create a new one as required, and enter the appropriate location for the job in the **Location** field.



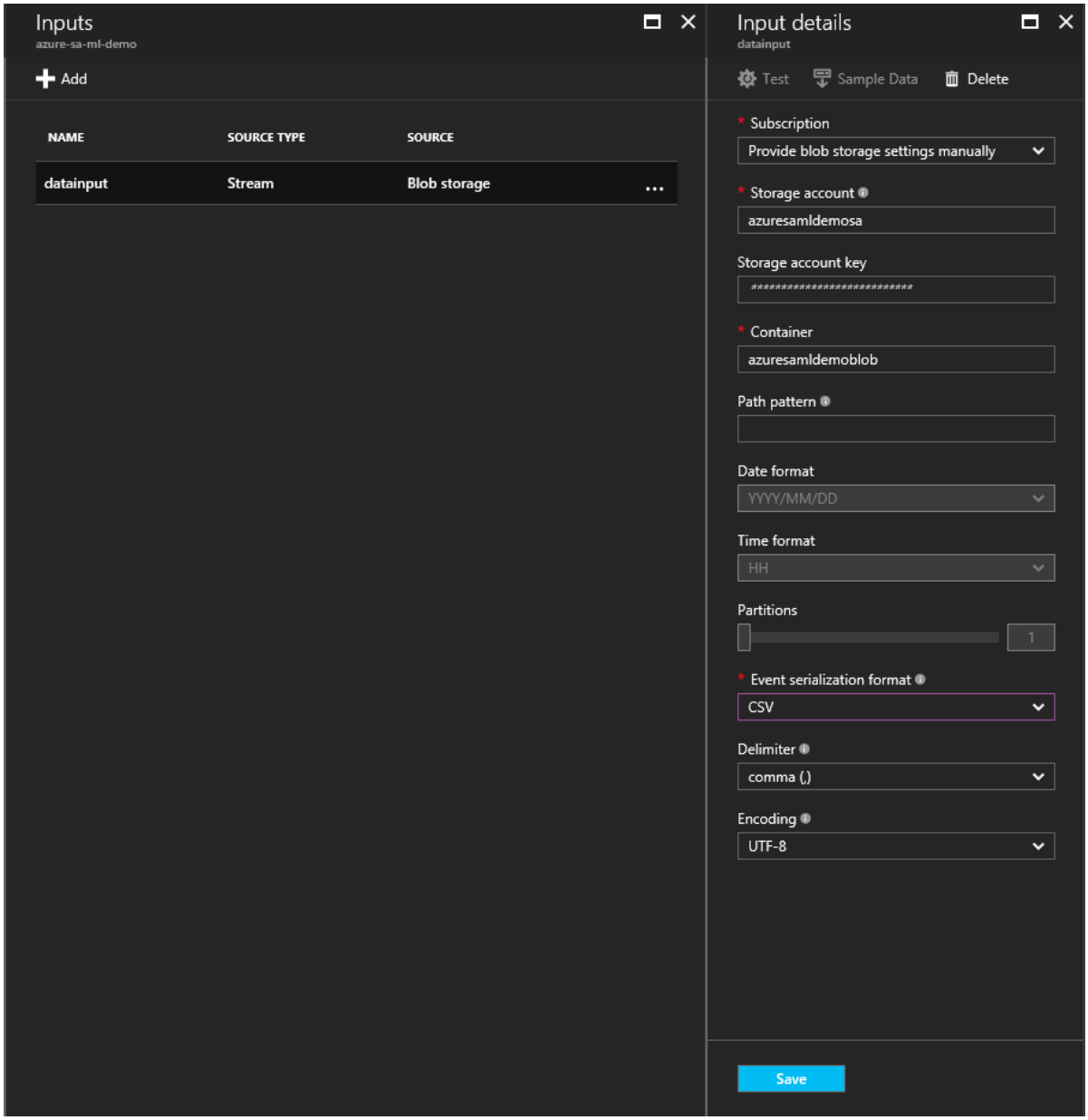
3. After the job is created, on the **Inputs** tab, select **Add an Input**.



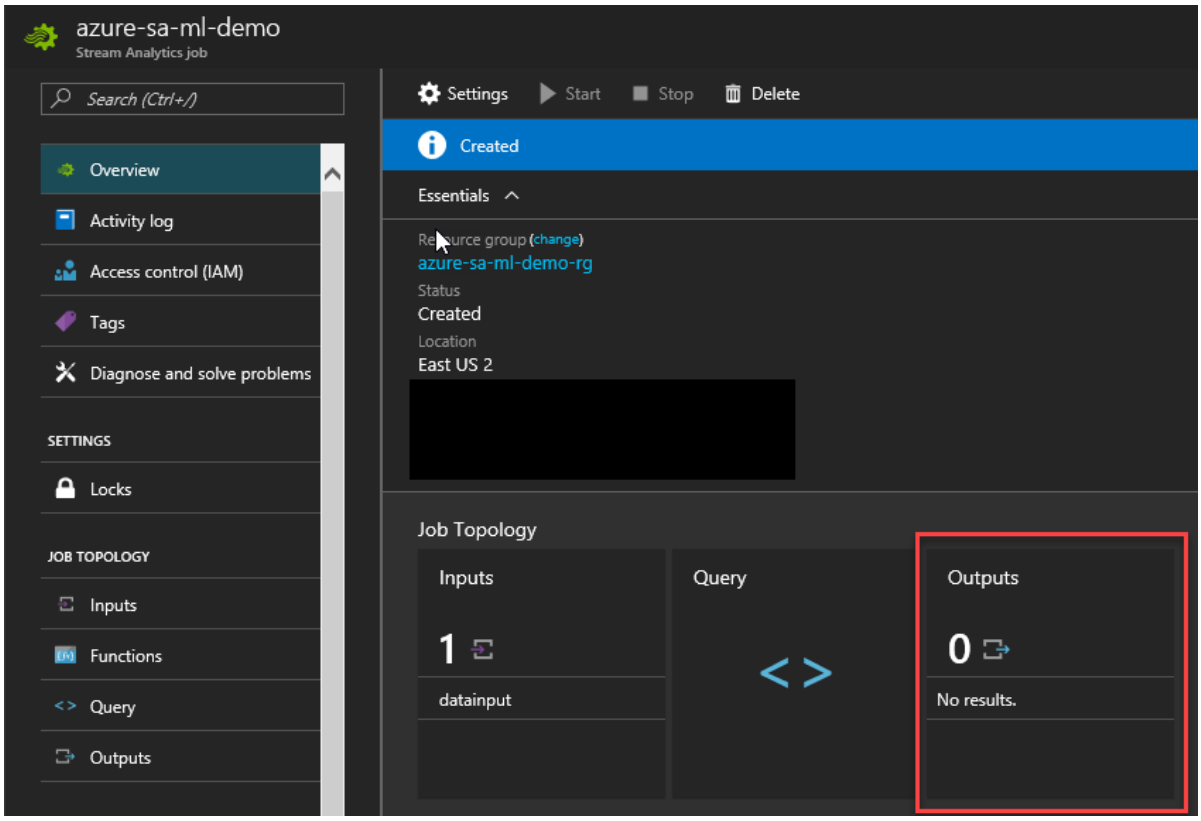
4. Select **Add** and then specify an **Input alias**, select **Data stream**, **Blob Storage** as the input, and then select **Next**.

5. On the **Blob Storage Settings** page of the wizard, provide the storage account blob container name you defined earlier when you uploaded the data. Click **Next**. For **Event Serialization Format**, select **CSV**.

Accept the default values for the rest of the **Serialization settings** page. Click **OK**.

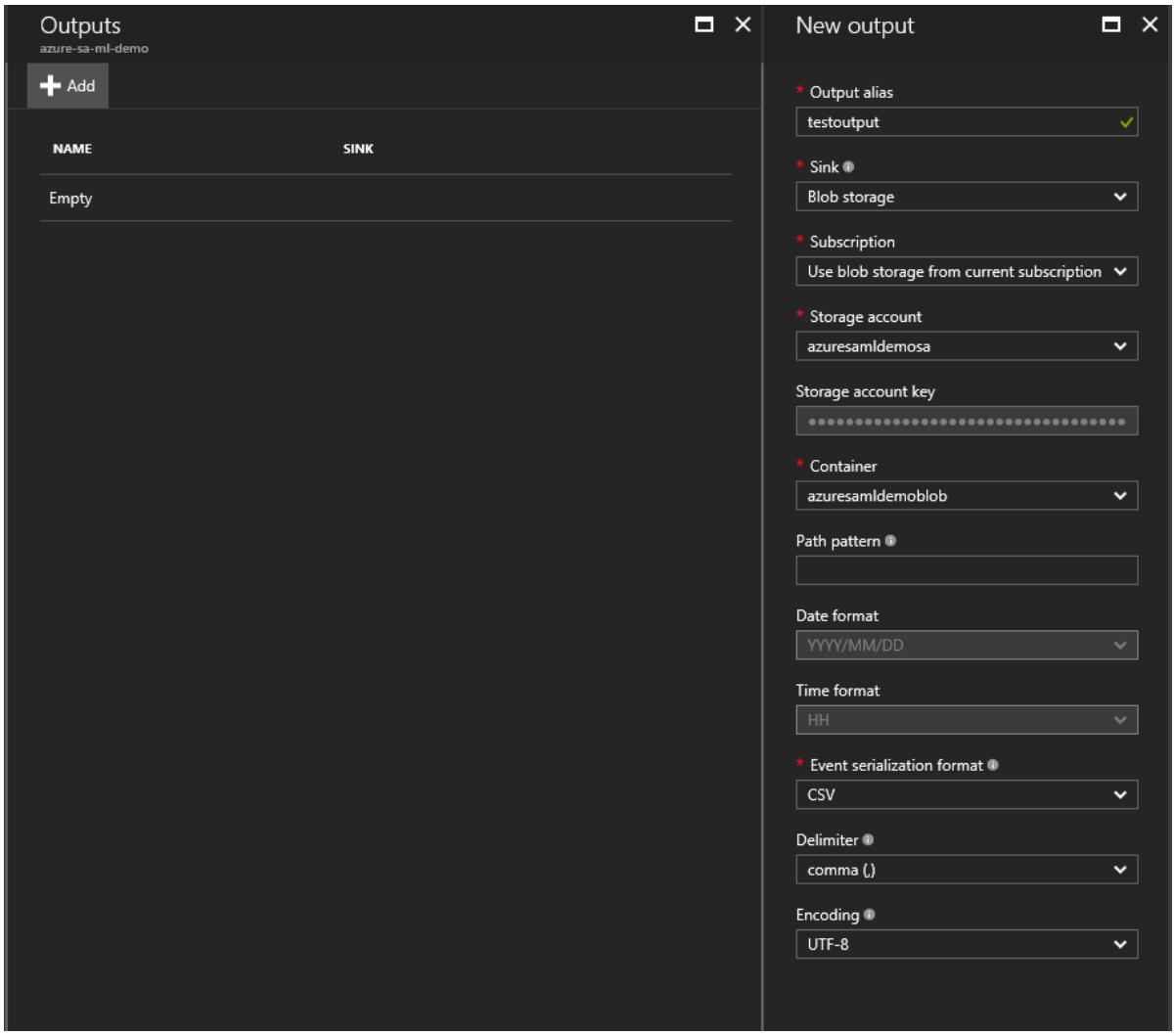


6. On the **Outputs** tab, select **Add an Output**.

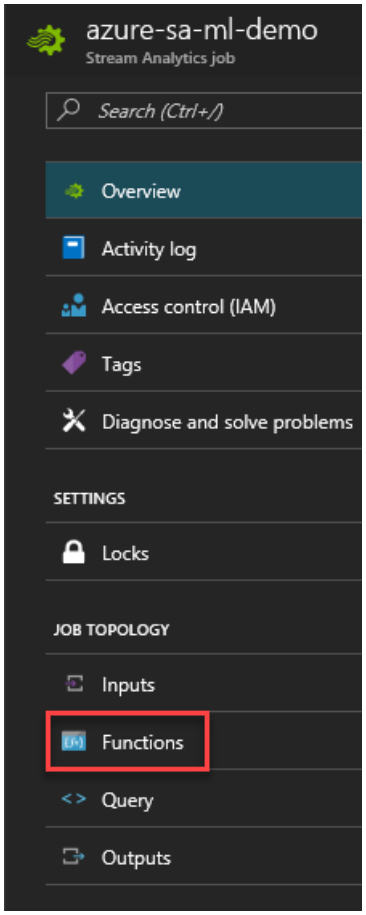


7. Click **Blob Storage**, and then enter the same parameters, except for the container. The value for **Input** was configured to read from the container named "test" where the **CSV** file was uploaded. For **Output**, enter "testoutput".

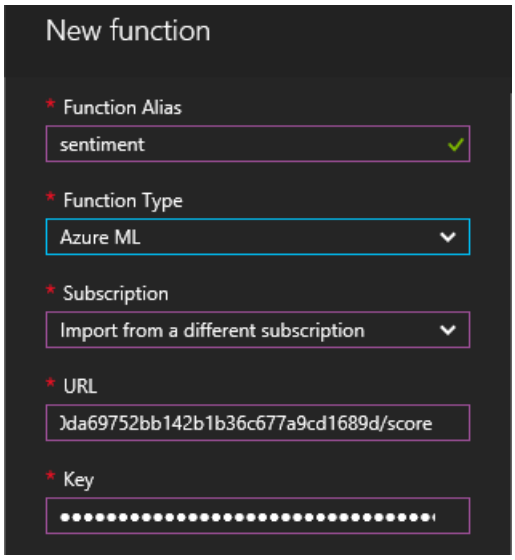
8. Validate that the output's **Serialization settings** are set to **CSV**, and then select the **OK** button.



9. On the **Functions** tab, select **Add a Machine Learning Function**.



10. On the **Machine Learning Web Service Settings** page, locate the Machine Learning workspace, web service, and default endpoint. For this article, apply the settings manually to gain familiarity with configuring a web service for any workspace, as long as you know the URL and have the API key. Enter the endpoint **URL** and **API key**. Click **OK**. Note that the **Function Alias** is 'sentiment'.



11. On the **Query** tab, modify the query as follows:

```
WITH sentiment AS (
  SELECT text, sentiment(text) as result from datainput
)

Select text, result.[Scored Labels]
Into testoutput
From sentiment
```

12. Click **Save** to save the query.

Start the Stream Analytics job and observe the output

1. Click **Start** at the top of the job page.
2. On the **Start Query Dialog**, select **Custom Time**, and then select one day prior to when you uploaded the CSV to Blob storage. Click **OK**.
3. Go to the Blob storage by using the tool you used to upload the CSV file, for example, Visual Studio.
4. A few minutes after the job is started, the output container is created and a CSV file is uploaded to it.
5. Open the file in the default CSV editor. Something similar to the following should be displayed:

text	scored probabilities
Azure is so cool	0.733864188
Cortana Analytics rocks	0.590195894
I hate going to Gym	0.403451264

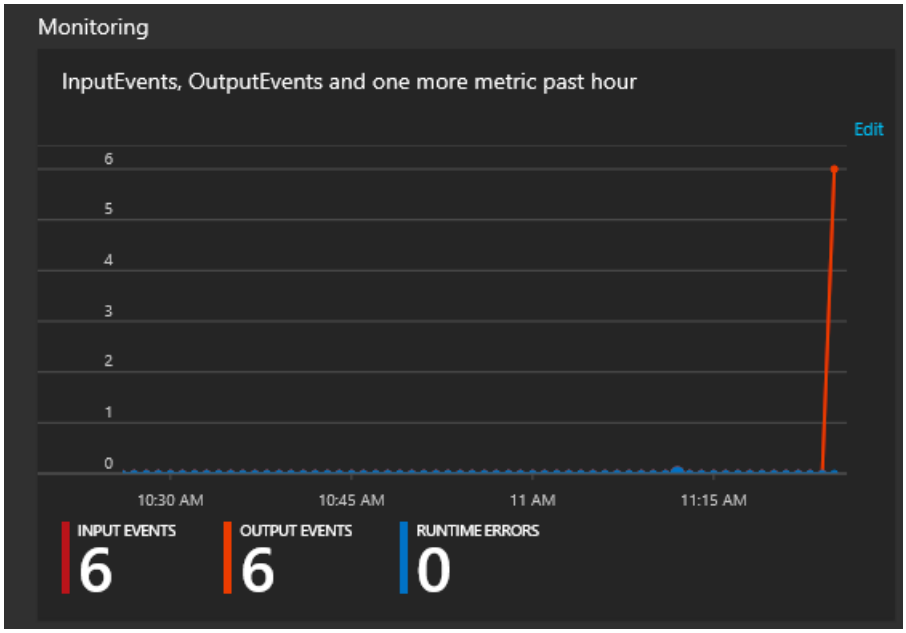
Conclusion

This article demonstrates how to create a Stream Analytics job that reads streaming text data and applies sentiment analytics to the data in real time. You can accomplish all of this without needing to worry about the intricacies of building a sentiment analytics model. This is one of the advantages of the Cortana Intelligence Suite.

You also can view Azure Machine Learning function-related metrics. To do this, select the **Monitor** tab. Three

function-related metrics are displayed.

- **Function Requests** indicates the number of requests sent to a Machine Learning web service.
- **Function Events** indicates the number of events in the request. By default, each request to a Machine Learning web service contains up to 1,000 events.



Reference architecture: Real-time event processing with Microsoft Azure Stream Analytics

1/25/2017 • 1 min to read • Edit on GitHub

The reference architecture for real-time event processing with Azure Stream Analytics is intended to provide a generic blueprint for deploying a real-time platform as a service (PaaS) stream-processing solution with Microsoft Azure.

Summary

Traditionally, analytics solutions have been based on capabilities such as ETL (extract, transform, load) and data warehousing, where data is stored prior to analysis. Changing requirements, including more rapidly arriving data, are pushing this existing model to the limit. The ability to analyze data within moving streams prior to storage is one solution, and while it is not a new capability, the approach has not been widely adopted across all industry verticals.

Microsoft Azure provides an extensive catalog of analytics technologies that are capable of supporting an array of different solution scenarios and requirements. Selecting which Azure services to deploy for an end-to-end solution can be a challenge given the breadth of offerings. This paper is designed to describe the capabilities and interoperation of the various Azure services that support an event-streaming solution. It also explains some of the scenarios in which customers can benefit from this type of approach.

Contents

- Executive Summary
- Introduction to Real-Time Analytics
- Value Proposition of Real-Time Data in Azure
- Common Scenarios for Real-Time Analytics
- Architecture and Components
  - Data Sources
  - Data-Integration Layer
  - Real-time Analytics Layer
  - Data Storage Layer
  - Presentation / Consumption Layer
- Conclusion

Author: Charles Feddersen, Solution Architect, Data Insights Center of Excellence, Microsoft Corporation

Published: January 2015

Revision: 1.0

Download: Real-Time Event Processing with Microsoft Azure Stream Analytics

Get help

For further assistance, try our Azure Stream Analytics forum

Next steps

- Introduction to Azure Stream Analytics
- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Query examples for common Stream Analytics usage patterns

1/25/2017 • 8 min to read • Edit on GitHub

Introduction

Queries in Azure Stream Analytics are expressed in a SQL-like query language, which is documented in the Stream Analytics Query Language Reference guide. This article outlines solutions to several common query patterns based on real world scenarios. It is a work in progress and will continue to be updated with new patterns on an ongoing basis.

Query example: Data type conversions

Description: Define the types of the properties on the input stream. For example, car weight is coming on the input stream as strings and needs to be converted to INT to perform SUM it up.

Input:

MAKE	TIME	WEIGHT
Honda	2015-01-01T00:00:01.0000000Z	"1000"
Honda	2015-01-01T00:00:02.0000000Z	"2000"

Output:

MAKE	WEIGHT
Honda	3000

Solution:

```
SELECT
    Make,
    SUM(CAST(Weight AS BIGINT)) AS Weight
FROM
    Input TIMESTAMP BY Time
GROUP BY
    Make,
    TumblingWindow(second, 10)
```

Explanation: Use a CAST statement on the Weight field to specify its type (see the list of supported Data Types here).

Query example: Using Like/Not like to do pattern matching

Description: Check that a field value on the event matches a certain pattern For example, return license plates that start with A and end with 9

Input:

MAKE	LICENSEPLATE	TIME
Honda	ABC-123	2015-01-01T00:00:01.0000000Z
Toyota	AAA-999	2015-01-01T00:00:02.0000000Z
Nissan	ABC-369	2015-01-01T00:00:03.0000000Z

Output:

MAKE	LICENSEPLATE	TIME
Toyota	AAA-999	2015-01-01T00:00:02.0000000Z
Nissan	ABC-369	2015-01-01T00:00:03.0000000Z

Solution:

```
SELECT
    *
FROM
    Input TIMESTAMP BY Time
WHERE
    LicensePlate LIKE 'A%9'
```

Explanation: Use the LIKE statement to check that the LicensePlate field value starts with A then has any string of zero or more characters and it ends with 9.

Query example: Specify logic for different cases/values (CASE statements)

Description: Provide different computation for a field based on some criteria. For example, provide a string description for how many cars passed of the same make with a special case for 1.

Input:

MAKE	TIME
Honda	2015-01-01T00:00:01.0000000Z
Toyota	2015-01-01T00:00:02.0000000Z
Toyota	2015-01-01T00:00:03.0000000Z

Output:

CARSPASSED	TIME
1 Honda	2015-01-01T00:00:10.0000000Z
2 Toyotas	2015-01-01T00:00:10.0000000Z

Solution:

```
SELECT
    CASE
        WHEN COUNT(*) = 1 THEN CONCAT('1 ', Make)
        ELSE CONCAT(CAST(COUNT(*) AS NVARCHAR(MAX)), ' ', Make, 's')
    END AS CarsPassed,
    System.TimeStamp AS Time
FROM
    Input TIMESTAMP BY Time
GROUP BY
    Make,
    TumblingWindow(second, 10)
```

Explanation: The CASE clause allows us to provide a different computation based on some criteria (in our case the count of cars in the aggregate window).

Query example: Send data to multiple outputs

Description: Send data to multiple output targets from a single job. For example, analyze data for a threshold-based alert and archive all events to blob storage

Input:

MAKE	TIME
Honda	2015-01-01T00:00:01.0000000Z
Honda	2015-01-01T00:00:02.0000000Z
Toyota	2015-01-01T00:00:01.0000000Z
Toyota	2015-01-01T00:00:02.0000000Z
Toyota	2015-01-01T00:00:03.0000000Z

Output1:

MAKE	TIME
Honda	2015-01-01T00:00:01.0000000Z
Honda	2015-01-01T00:00:02.0000000Z
Toyota	2015-01-01T00:00:01.0000000Z
Toyota	2015-01-01T00:00:02.0000000Z
Toyota	2015-01-01T00:00:03.0000000Z

Output2:

MAKE	TIME	COUNT
Toyota	2015-01-01T00:00:10.0000000Z	3

Solution:

```
SELECT
    *
INTO
    ArchiveOutput
FROM
    Input TIMESTAMP BY Time

SELECT
    Make,
    System.TimeStamp AS Time,
    COUNT(*) AS [Count]
INTO
    AlertOutput
FROM
    Input TIMESTAMP BY Time
GROUP BY
    Make,
    TumblingWindow(second, 10)
HAVING
    [Count] >= 3
```

Explanation: The INTO clause tells Stream Analytics which of the outputs to write the data from this statement. The first query is a pass-through of the data we received to an output that we named ArchiveOutput. The second query does some simple aggregation and filtering and sends the results to a downstream alerting system. *Note*: You can also reuse results of CTEs (i.e. WITH statements) in multiple output statements – this has the added benefit of opening fewer readers to the input source. For example,

```
WITH AllRedCars AS (
    SELECT
        *
    FROM
        Input TIMESTAMP BY Time
    WHERE
        Color = 'red'
)
SELECT * INTO HondaOutput FROM AllRedCars WHERE Make = 'Honda'
SELECT * INTO ToyotaOutput FROM AllRedCars WHERE Make = 'Toyota'
```

Query example: Counting unique values

Description: count the number of unique field values that appear in the stream within a time window. For example, how many unique make of cars passed through the toll booth in a 2 second window?

Input:

MAKE	TIME
Honda	2015-01-01T00:00:01.0000000Z
Honda	2015-01-01T00:00:02.0000000Z
Toyota	2015-01-01T00:00:01.0000000Z
Toyota	2015-01-01T00:00:02.0000000Z
Toyota	2015-01-01T00:00:03.0000000Z

Output:

COUNT	TIME
2	2015-01-01T00:00:02.000Z
1	2015-01-01T00:00:04.000Z

Solution:

```
SELECT
     COUNT(DISTINCT Make) AS CountMake,
     System.TIMESTAMP AS TIME
FROM Input TIMESTAMP BY TIME
GROUP BY
     TumblingWindow(second, 2)
```

Explanation: COUNT(DISTINCT Make) returns the number of distinct values of the "Make" column within a time window.

Query example: Determine if a value has changed

Description: Look at a previous value to determine if it is different than the current value For example, is the previous car on the Toll Road the same make as the current car?

Input:

MAKE	TIME
Honda	2015-01-01T00:00:01.0000000Z
Toyota	2015-01-01T00:00:02.0000000Z

Output:

MAKE	TIME
Toyota	2015-01-01T00:00:02.0000000Z

Solution:

```
SELECT
    Make,
    Time
FROM
    Input TIMESTAMP BY Time
WHERE
    LAG(Make, 1) OVER (LIMIT DURATION(minute, 1)) <> Make
```

Explanation: Use LAG to peek into the input stream one event back and get the Make value. Then compare it to the Make on the current event and output the event if they are different.

Query example: Find first event in a window

Description: Find first car in every 10 minute interval?

Input:

LICENSEPLATE	MAKE	TIME
DXE 5291	Honda	2015-07-27T00:00:05.0000000Z
YZK 5704	Ford	2015-07-27T00:02:17.0000000Z
RMV 8282	Honda	2015-07-27T00:05:01.0000000Z
YHN 6970	Toyota	2015-07-27T00:06:00.0000000Z
VFE 1616	Toyota	2015-07-27T00:09:31.0000000Z
QYF 9358	Honda	2015-07-27T00:12:02.0000000Z
MDR 6128	BMW	2015-07-27T00:13:45.0000000Z

Output:

LICENSEPLATE	MAKE	TIME
DXE 5291	Honda	2015-07-27T00:00:05.0000000Z
QYF 9358	Honda	2015-07-27T00:12:02.0000000Z

Solution:

```
SELECT
    LicensePlate,
    Make,
    Time
FROM
    Input TIMESTAMP BY Time
WHERE
    IsFirst(minute, 10) = 1
```

Now let's change the problem and find first car of particular Make in every 10 minute interval.

LICENSEPLATE	MAKE	TIME
DXE 5291	Honda	2015-07-27T00:00:05.0000000Z
YZK 5704	Ford	2015-07-27T00:02:17.0000000Z
YHN 6970	Toyota	2015-07-27T00:06:00.0000000Z
QYF 9358	Honda	2015-07-27T00:12:02.0000000Z
MDR 6128	BMW	2015-07-27T00:13:45.0000000Z

Solution:

```
SELECT
    LicensePlate,
    Make,
    Time
FROM
    Input TIMESTAMP BY Time
WHERE
    IsFirst(minute, 10) OVER (PARTITION BY Make) = 1
```

Query example: Find last event in a window

Description: Find last car in every 10 minute interval.

Input:

LICENSEPLATE	MAKE	TIME
DXE 5291	Honda	2015-07-27T00:00:05.0000000Z
YZK 5704	Ford	2015-07-27T00:02:17.0000000Z
RMV 8282	Honda	2015-07-27T00:05:01.0000000Z
YHN 6970	Toyota	2015-07-27T00:06:00.0000000Z
VFE 1616	Toyota	2015-07-27T00:09:31.0000000Z
QYF 9358	Honda	2015-07-27T00:12:02.0000000Z
MDR 6128	BMW	2015-07-27T00:13:45.0000000Z

Output:

LICENSEPLATE	MAKE	TIME
VFE 1616	Toyota	2015-07-27T00:09:31.0000000Z
MDR 6128	BMW	2015-07-27T00:13:45.0000000Z

Solution:

```
WITH LastInWindow AS
(
    SELECT
        MAX(Time) AS LastEventTime
    FROM
        Input TIMESTAMP BY Time
    GROUP BY
        TumblingWindow(minute, 10)
)
SELECT
    Input.LicensePlate,
    Input.Make,
    Input.Time
FROM
    Input TIMESTAMP BY Time
    INNER JOIN LastInWindow
    ON DATEDIFF(minute, Input, LastInWindow) BETWEEN 0 AND 10
    AND Input.Time = LastInWindow.LastEventTime
```

Explanation: There are two steps in the query – the first one finds latest timestamp in 10 minute windows. The second step joins results of the first query with original stream to find events matching last timestamps in each window.

Query example: Detect the absence of events

Description: Check that a stream has no value that matches a certain criteria. For example, have 2 consecutive cars from the same make entered the toll road within 90 seconds?

Input:

MAKE	LICENSEPLATE	TIME
Honda	ABC-123	2015-01-01T00:00:01.0000000Z
Honda	AAA-999	2015-01-01T00:00:02.0000000Z
Toyota	DEF-987	2015-01-01T00:00:03.0000000Z
Honda	GHI-345	2015-01-01T00:00:04.0000000Z

Output:

MAKE	TIME	CURRENTCARLICENSEPLATE	FIRSTCARLICENSEPLATE	FIRSTCARTIME
Honda	2015-01-01T00:00:02.0000000Z	AAA-999	ABC-123	2015-01-01T00:00:01.0000000Z

Solution:

```
SELECT
    Make,
    Time,
    LicensePlate AS CurrentCarLicensePlate,
    LAG(LicensePlate, 1) OVER (LIMIT DURATION(second, 90)) AS FirstCarLicensePlate,
    LAG(Time, 1) OVER (LIMIT DURATION(second, 90)) AS FirstCarTime
FROM
    Input TIMESTAMP BY Time
WHERE
    LAG(Make, 1) OVER (LIMIT DURATION(second, 90)) = Make
```

Explanation: Use LAG to peek into the input stream one event back and get the Make value. Then compare it to the Make on the current event and output the event if they are the same and use LAG to get data about the previous car.

Query example: Detect duration between events

Description: Find the duration of a given event. For example, given a web clickstream determine time spent on a feature.

Input:

USER	FEATURE	EVENT	TIME
user@location.com	RightMenu	Start	2015-01-01T00:00:01.0000000Z
user@location.com	RightMenu	End	2015-01-01T00:00:08.0000000Z

Output:

USER	FEATURE	DURATION
user@location.com	RightMenu	7

Solution

```
    SELECT
        [user], feature, DATEDIFF(second, LAST(Time) OVER (PARTITION BY [user], feature LIMIT DURATION(hour,
1) WHEN Event = 'start'), Time) as duration
    FROM input TIMESTAMP BY Time
    WHERE
        Event = 'end'
```

Explanation: Use LAST function to retrieve last Time value when event type was 'Start'. Note that LAST function uses PARTITION BY [user] to indicate that result shall be computed per unique user. The query has a 1 hour maximum threshold for time difference between 'Start' and 'Stop' events but is configurable as needed (LIMIT DURATION(hour, 1).

Query example: Detect duration of a condition

Description: Find out how long a condition occurred for. For example, suppose that a bug that resulted in all cars having an incorrect weight (above 20,000 pounds) – we want to compute the duration of the bug.

Input:

MAKE	TIME	WEIGHT
Honda	2015-01-01T00:00:01.0000000Z	2000
Toyota	2015-01-01T00:00:02.0000000Z	25000
Honda	2015-01-01T00:00:03.0000000Z	26000
Toyota	2015-01-01T00:00:04.0000000Z	25000
Honda	2015-01-01T00:00:05.0000000Z	26000
Toyota	2015-01-01T00:00:06.0000000Z	25000
Honda	2015-01-01T00:00:07.0000000Z	26000
Toyota	2015-01-01T00:00:08.0000000Z	2000

Output:

STARTFAULT	ENDFAULT
2015-01-01T00:00:02.000Z	2015-01-01T00:00:07.000Z

Solution:

```
WITH SelectPreviousEvent AS
(
SELECT
*,
    LAG([time]) OVER (LIMIT DURATION(hour, 24)) as previousTime,
    LAG([weight]) OVER (LIMIT DURATION(hour, 24)) as previousWeight
FROM input TIMESTAMP BY [time]
)

SELECT
    LAG(time) OVER (LIMIT DURATION(hour, 24) WHEN previousWeight < 20000 ) [StartFault],
    previousTime [EndFault]
FROM SelectPreviousEvent
WHERE
    [weight] < 20000
    AND previousWeight > 20000
```

Explanation: Use LAG to view the input stream for 24 hours and look for instances where StartFault and StopFault are spanned by weight < 20000.

Query example: Fill missing values

Description: For the stream of events that have missing values, produce a stream of events with regular intervals. For example, generate event every 5 seconds that will report the most recently seen data point.

Input:

T	VALUE
"2014-01-01T06:01:00"	1

T	VALUE
"2014-01-01T06:01:05"	2
"2014-01-01T06:01:10"	3
"2014-01-01T06:01:15"	4
"2014-01-01T06:01:30"	5
"2014-01-01T06:01:35"	6

Output (first 10 rows):

WINDOWEND	LASTEVENT.T	LASTEVENT.VALUE
2014-01-01T14:01:00.000Z	2014-01-01T14:01:00.000Z	1
2014-01-01T14:01:05.000Z	2014-01-01T14:01:05.000Z	2
2014-01-01T14:01:10.000Z	2014-01-01T14:01:10.000Z	3
2014-01-01T14:01:15.000Z	2014-01-01T14:01:15.000Z	4
2014-01-01T14:01:20.000Z	2014-01-01T14:01:15.000Z	4
2014-01-01T14:01:25.000Z	2014-01-01T14:01:15.000Z	4
2014-01-01T14:01:30.000Z	2014-01-01T14:01:30.000Z	5
2014-01-01T14:01:35.000Z	2014-01-01T14:01:35.000Z	6
2014-01-01T14:01:40.000Z	2014-01-01T14:01:35.000Z	6
2014-01-01T14:01:45.000Z	2014-01-01T14:01:35.000Z	6

Solution:

```
SELECT
    System.Timestamp AS windowEnd,
    TopOne() OVER (ORDER BY t DESC) AS lastEvent
FROM
    input TIMESTAMP BY t
GROUP BY HOPPINGWINDOW(second, 300, 5)
```

Explanation: This query will generate events every 5 second and will output the last event that was received before. Hopping Window duration determines how far back the query will look to find the latest event (300 seconds in this example).

Get help

For further assistance, try our Azure Stream Analytics forum

Next steps

- Introduction to Azure Stream Analytics
- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Use Azure Stream Analytics Tool for Visual Studio

2/1/2017 • 6 min to read • Edit on GitHub

Azure Stream Analytics tools for Visual Studio are now generally available. These tools enable a richer experience for Stream Analytics user to troubleshoot as well as write complex queries and even write queries locally. You will also have the ability to export a Stream Analytics job into a Visual Studio project.

Introduction

In this tutorial, you will learn how to use Azure Stream Analytics Tools for Visual Studio to create, author, test locally, manage and debug your Azure Stream Analytics jobs.

After completing this tutorial, you will be able to:

- Familiarize yourself with the Azure Stream Analytics Tools for Visual Studio.
- Configure and deploy a Stream Analytics job.
- Test your job locally with local sample data.
- Use the monitoring to troubleshoot issues.
- Export existing jobs to projects.

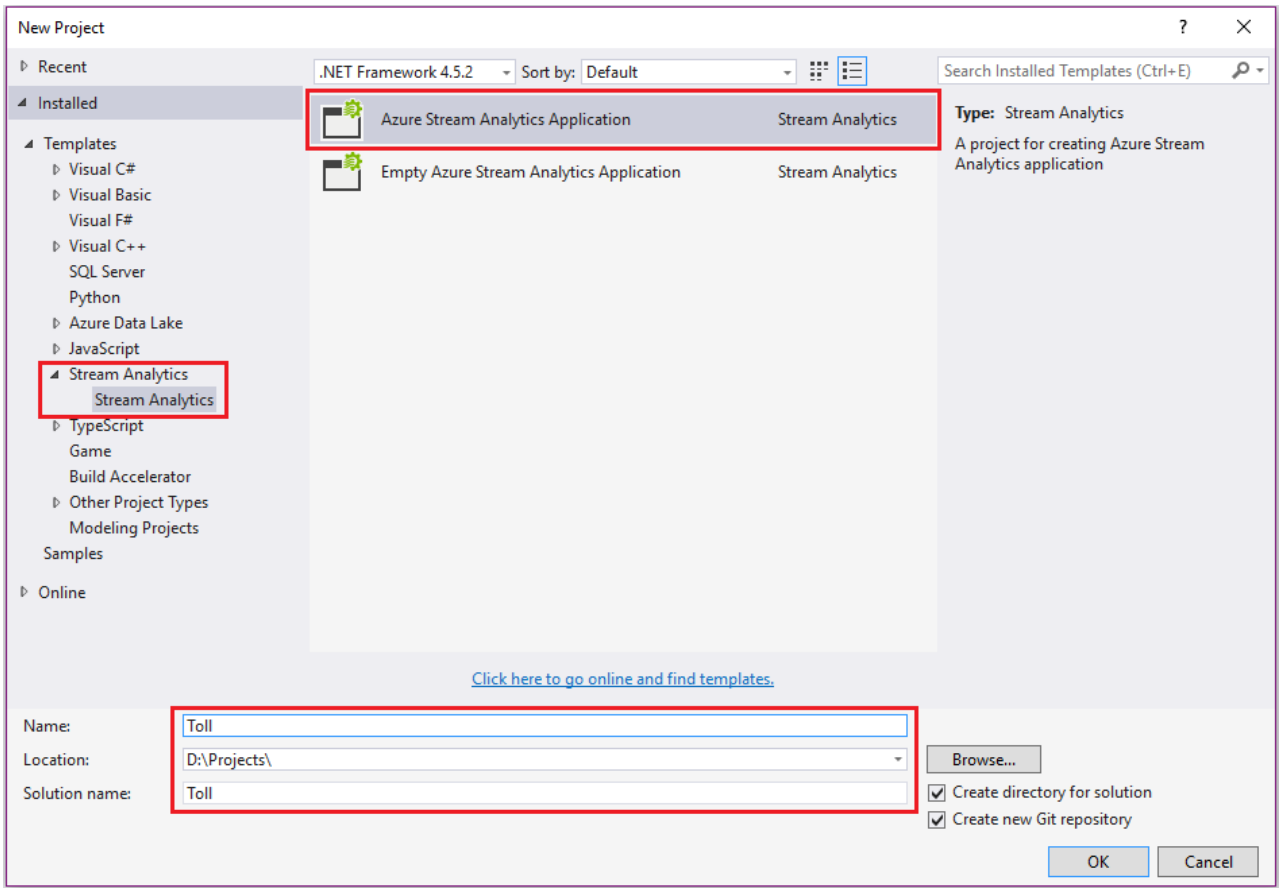
Prerequisites

You will need the following prerequisites to complete this tutorial:

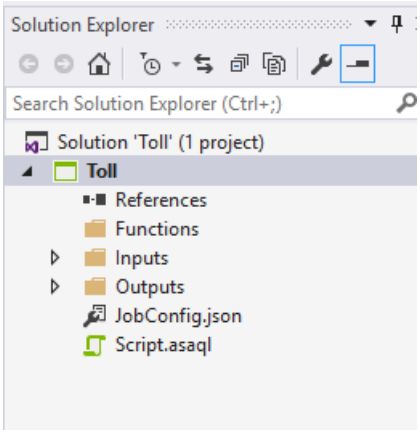
- Finish the steps before **Create a Stream Analytics job** from the Build an IoT solution by using Stream Analytics tutorial.
- Visual Studio 2015, Visual Studio 2013 update 4, or Visual Studio 2012. Enterprise (Ultimate/Premium), Professional, Community editions are supported; Express edition is not supported. Visual Studio 2017 is currently not supported.
- Microsoft Azure SDK for .NET version 2.7.1 or above. Install it using the Web platform installer.
- Installation of Azure Stream Analytics Tools for Visual Studio.

Create a Stream Analytics Project

In Visual Studio, click the **File Menu** and choose **New Project**. Choose **Stream Analytics** from the templates list on the left and then click **Azure Stream Analytics Application**. Input the Project Name, Location and Solution name at the bottom as you do for other projects.



You will see a project **Toll** generated in **Solution Explorer**.



Choose the correct Subscription

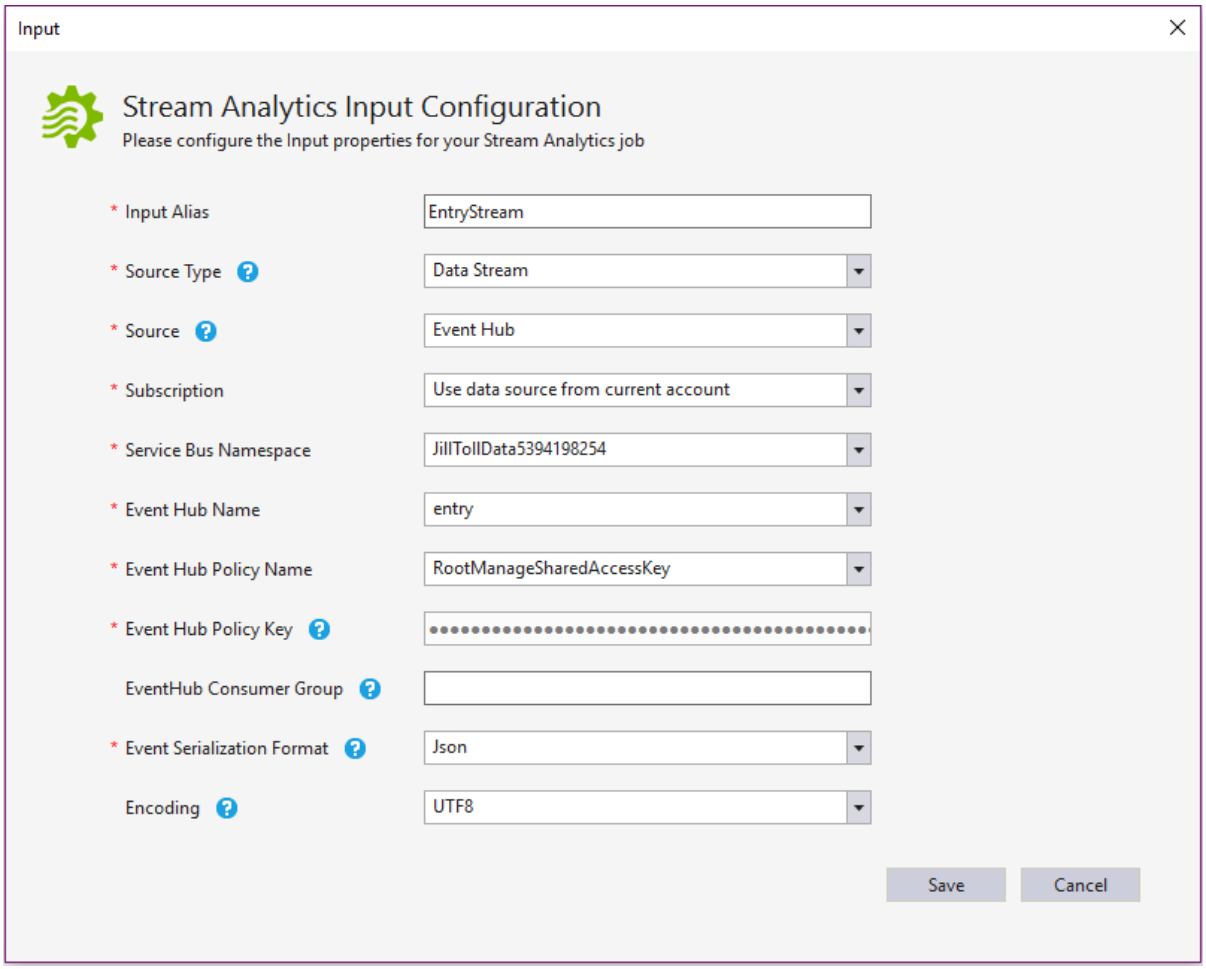
1. Open **Server Explorer** in Visual Studio from **View** menu.
2. Log in with your Azure Account.

Define input sources

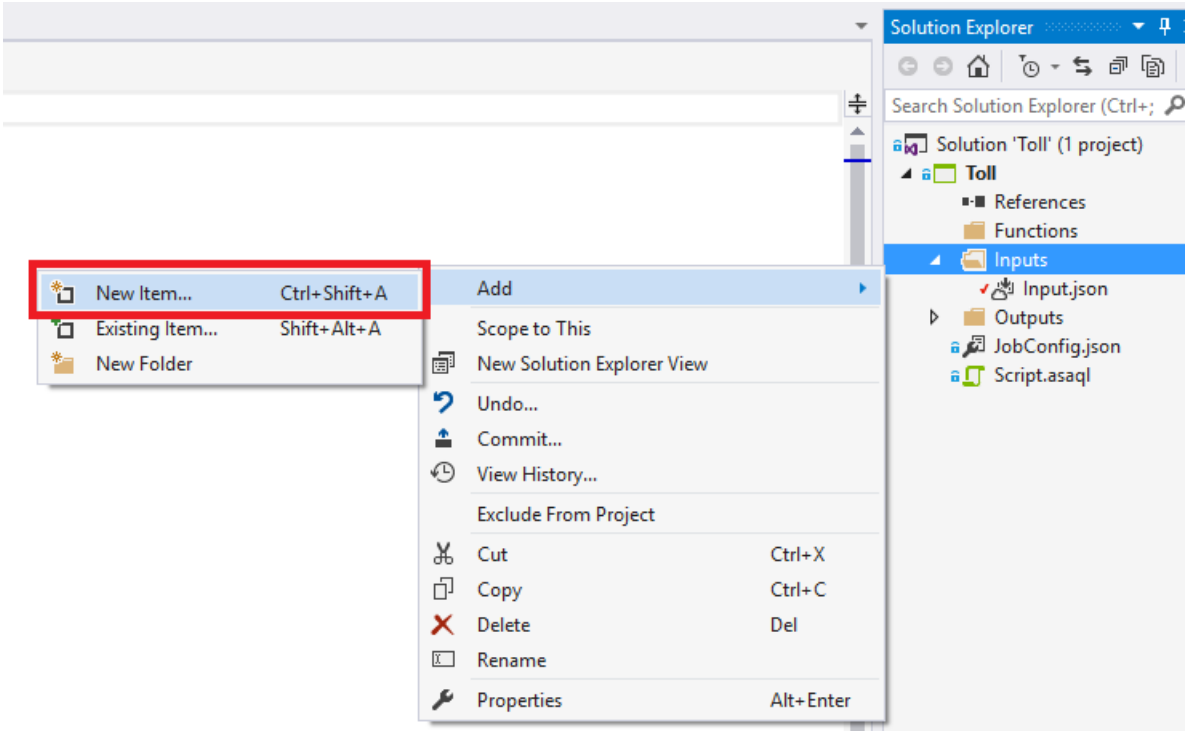
1. In **Solution Explorer**, expand **Inputs** node and rename **Input.json** to **EntryStream.json**. Double-click **EntryStream.json**.
2. Your **INPUT ALIAS** now should be **EntryStream**. Please note that input alias is the one will be used in query script.
3. Source Type is **Data Stream**.
4. Source is **Event hub**.
5. Service Bus Namescape should be the **tollData** one in the drop-down.
6. Event hub name should be set to **entry**.

7. Event hub policy name is **RootManageSharedAccessKey** (the default value).
8. Select **JSON** for **EVENT SERIALIZATION FORMAT** and **UTF8** for **ENCODING**.

   Your settings will look like:

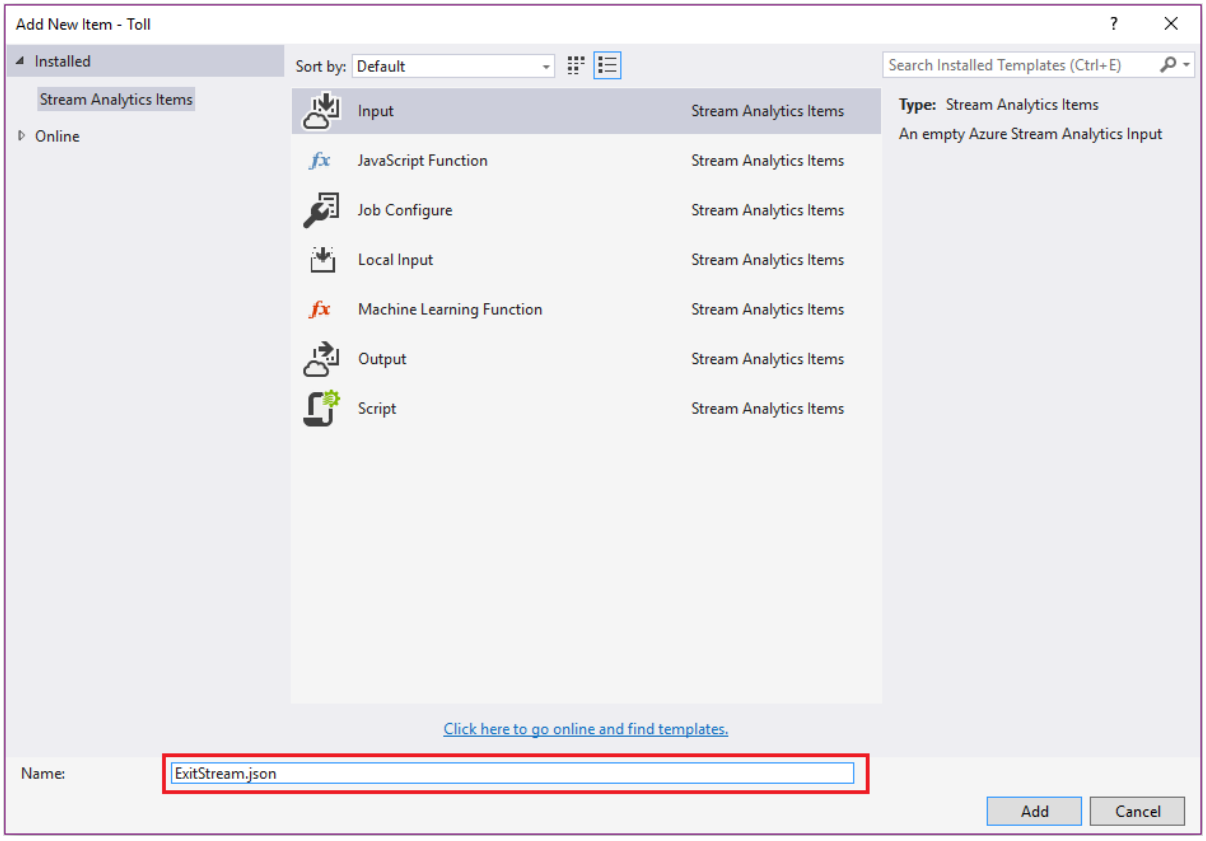


9. Click **Save** at the bottom of the page to finish the wizard. Now you can add another input source to create the exit stream. Right-click the inputs node and click **New Item**.

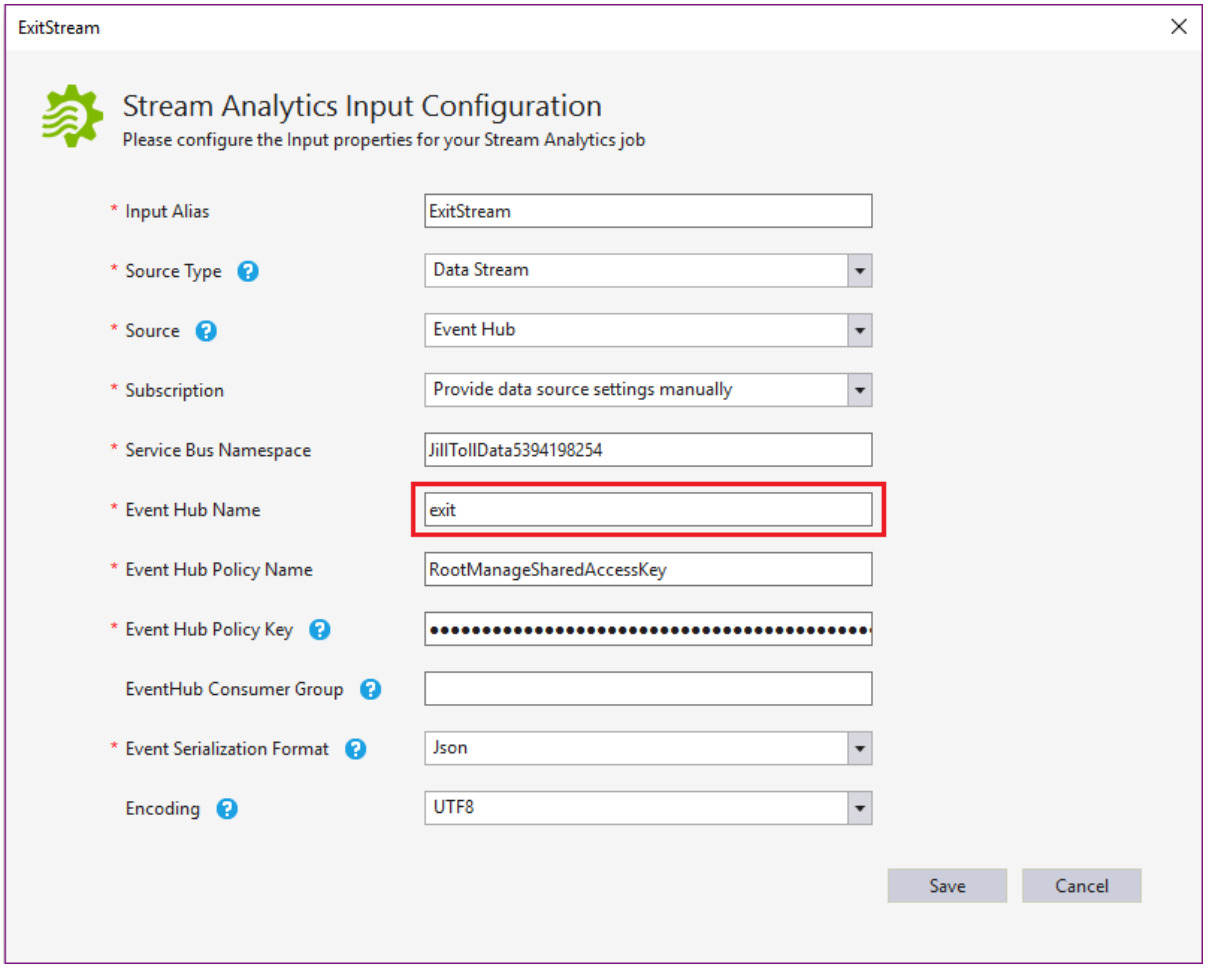


10. In the popped up window, choose **Azure Stream Analytics Input** and change the Name to

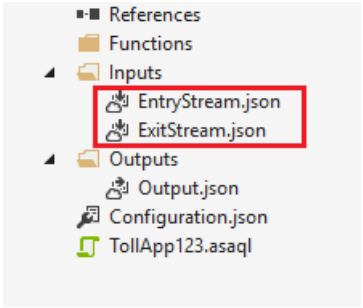
ExitStream.json. Click **Add**.



11. Double-click **ExitStream.json** in the project and follow the same steps as the entry stream to fill in. Be sure to enter values for Event Hub name as on the following screenshot.

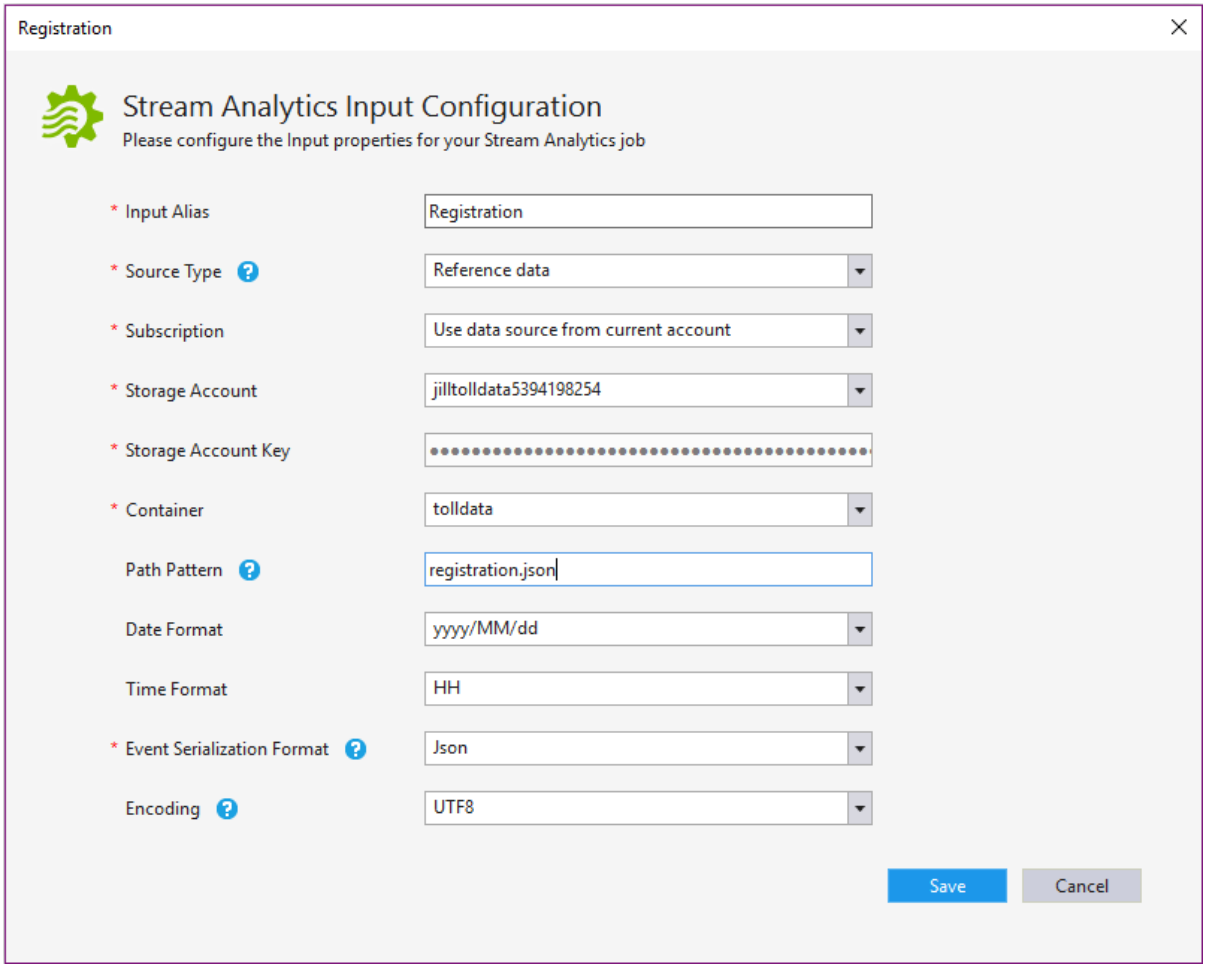


Now you have defined two input streams.



Next, you will add reference data input for the blob file that contains car registration data.

12. Right-click the **Inputs** node in the project, and then follow the same process for the stream inputs but select **REFERENCE DATA** instead of Data Stream and the Input Alias is **Registration**.



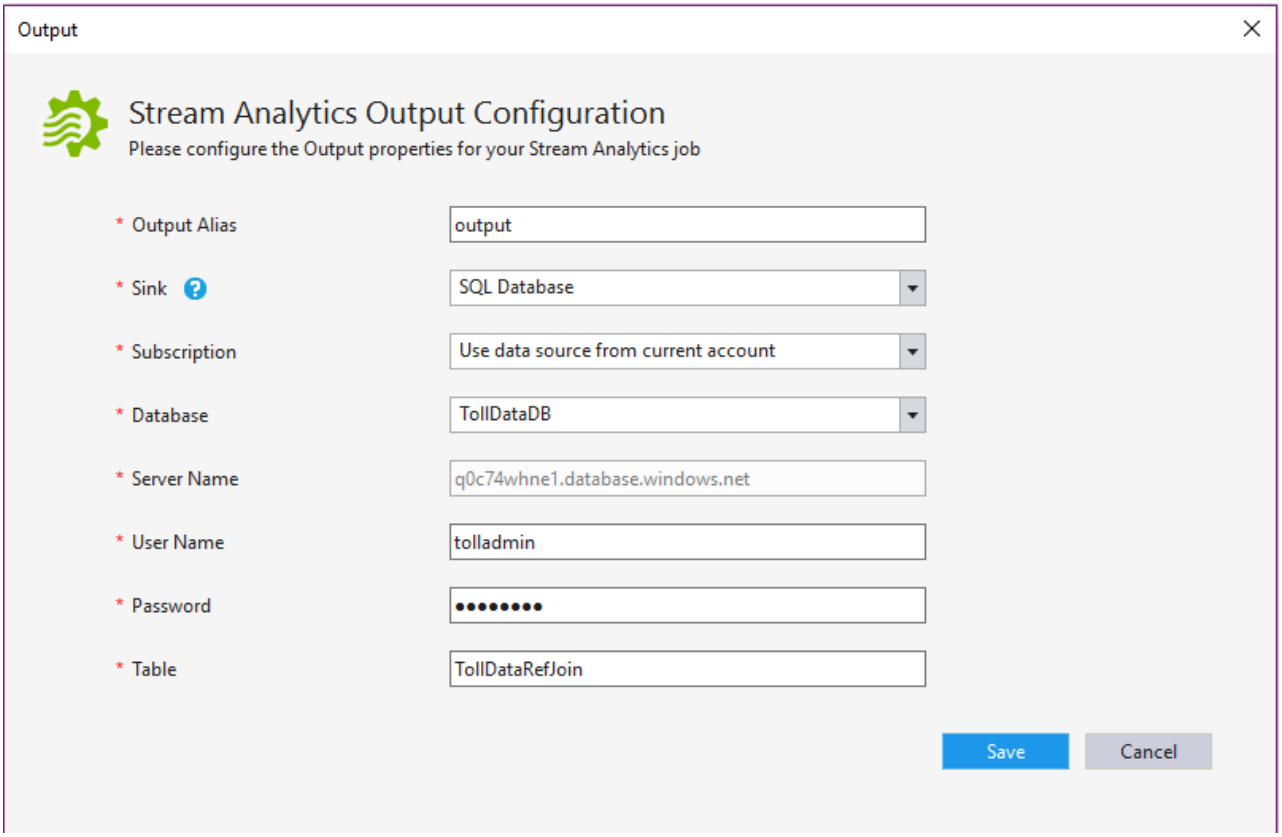
13. Select Storage account that contains with **tolldata**. The container name should be **tolldata**, and the **PATH PATTERN** should be **registration.json**. This file name is case-sensitive and should be lowercase.

14. Click **Save** to finish the wizard.

Now all inputs are defined.

Define output

1. In **Solution Explorer**, expand **Inputs** node and double-click **Output.json**.
2. Set the Output alias to **output** and then Sink to SQL database.
3. Enter the database name: **TollDataDB**.
4. Enter **tolladmin** in the **USERNAME** field, **123toll!** in the **PASSWORD** field, and **TollDataRefJoin** in the **TABLE** field.
5. Click **Save**.



Azure Stream Analytics Query

This tutorial attempts to answer several business questions that are related to toll data and constructs Stream Analytics queries that can be used in Azure Stream Analytics to provide a relevant answer. Before you start your first Stream Analytics job, let's explore a simple scenarios and the query syntax.

Introduction to Azure Stream Analytics query language

Let's say that you need to count the number of vehicles that enter a toll booth. Because this is a continuous stream of events, you have to define a "period of time." Let's modify the question to be "How many vehicles enter a toll booth every three minutes?". This is commonly referred to as the tumbling count.

Let's look at the Azure Stream Analytics query that answers this question:

```
SELECT TollId, System.Timestamp AS WindowEnd, COUNT(*) AS Count
FROM EntryStream TIMESTAMP BY EntryTime
GROUP BY TUMBLINGWINDOW(minute, 3), TollId
```

As you can see, Azure Stream Analytics uses a query language that's like SQL and adds a few extensions to specify time-related aspects of the query.

For more details, read about Time Management and Windowing constructs used in the query from MSDN.

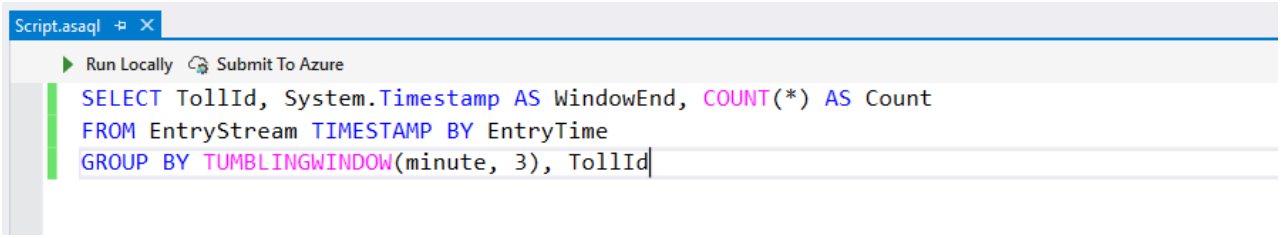
Now that you have written your first Azure Stream Analytics query, it is time to test it by using sample data files located in your TollApp folder in the following path:

..\TollApp\TollApp\Data

This folder contains the following files: • Entry.json • Exit.json • Registration.json

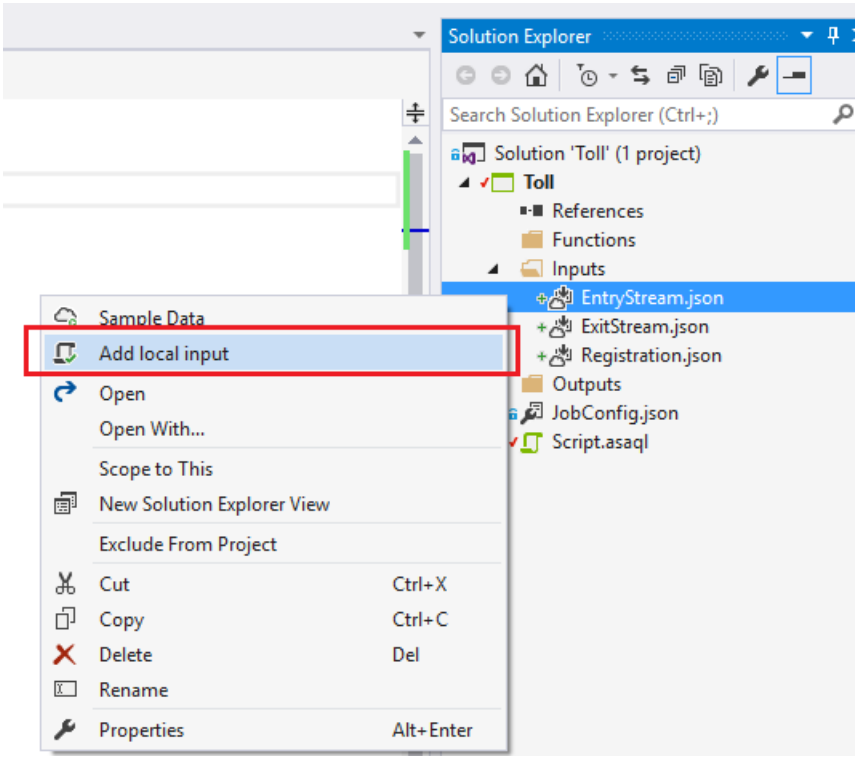
Question: Number of vehicles entering a toll booth

In the project, double-click Script.asaql to open the script in editor and paste the script in previous section into the editor. The query editor supports Intellisense, syntax coloring and Error marker.

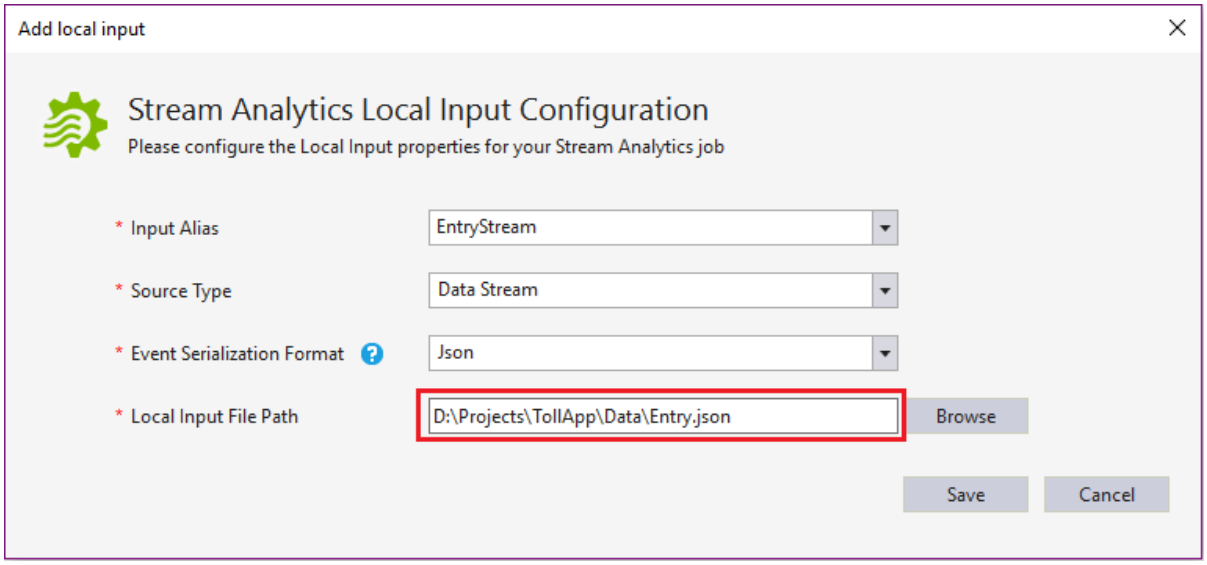


Testing Azure Stream Analytics queries locally

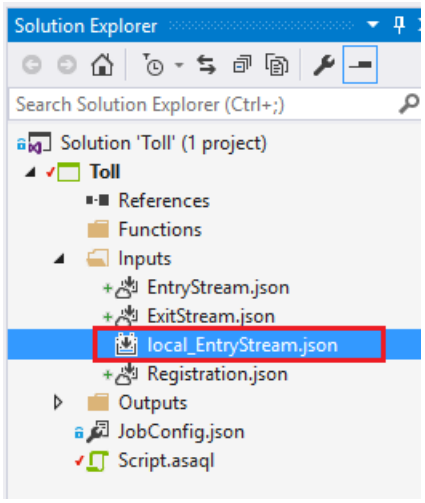
1. You can first compile the query to see if there is any syntax error. [TBD]
2. To validate this query against sample data, you can use local sample data by right-clicking the input and select **Add local input** from the context menu.



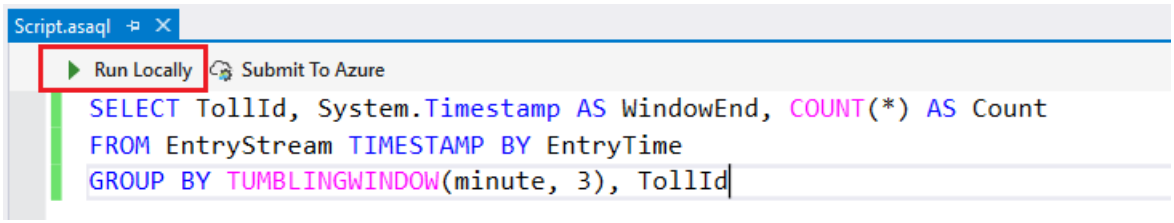
In the pop-up window select the sample data from your local path. Click **Save**.



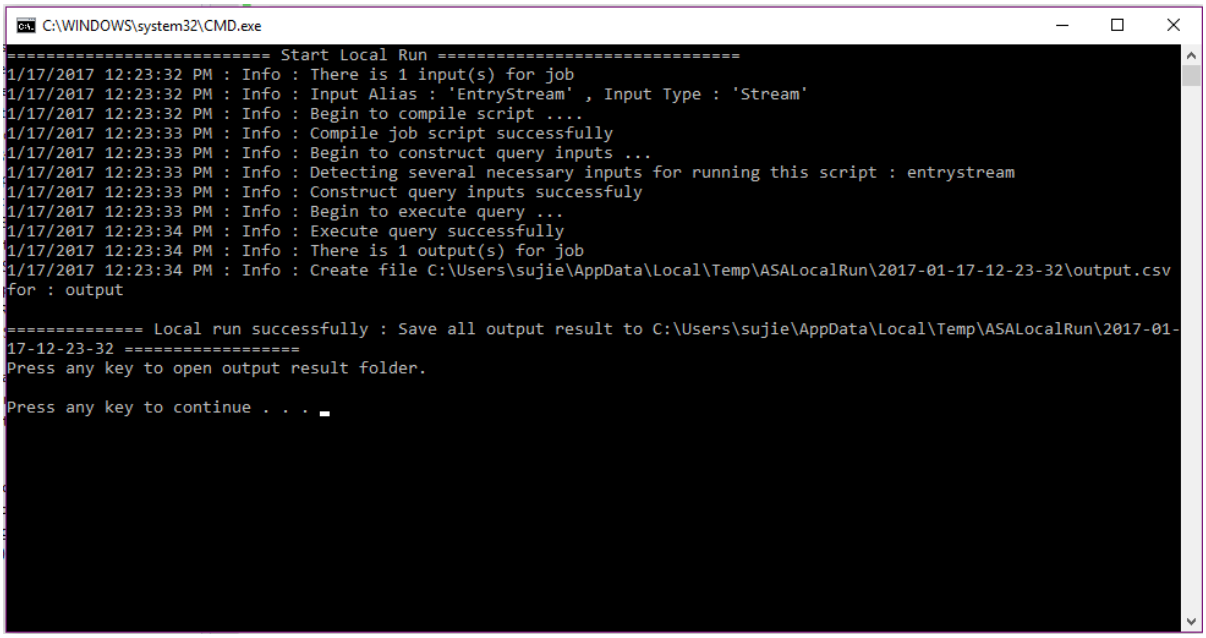
A file named **local_EntryStream.json** will be added automatically to your inputs folder.



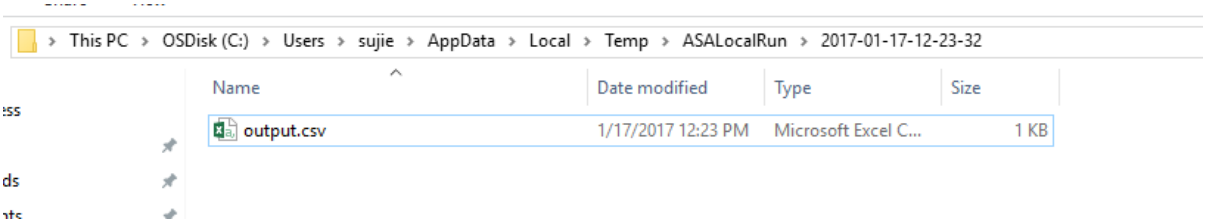
3. Click Run Locally in query editor. Or you can press F5.



You can find output path from console output and press any key to open the result folder.

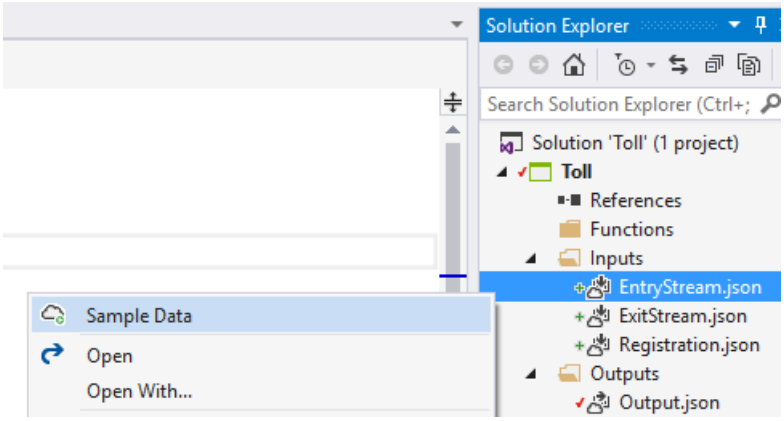


4. Check result in local folder.

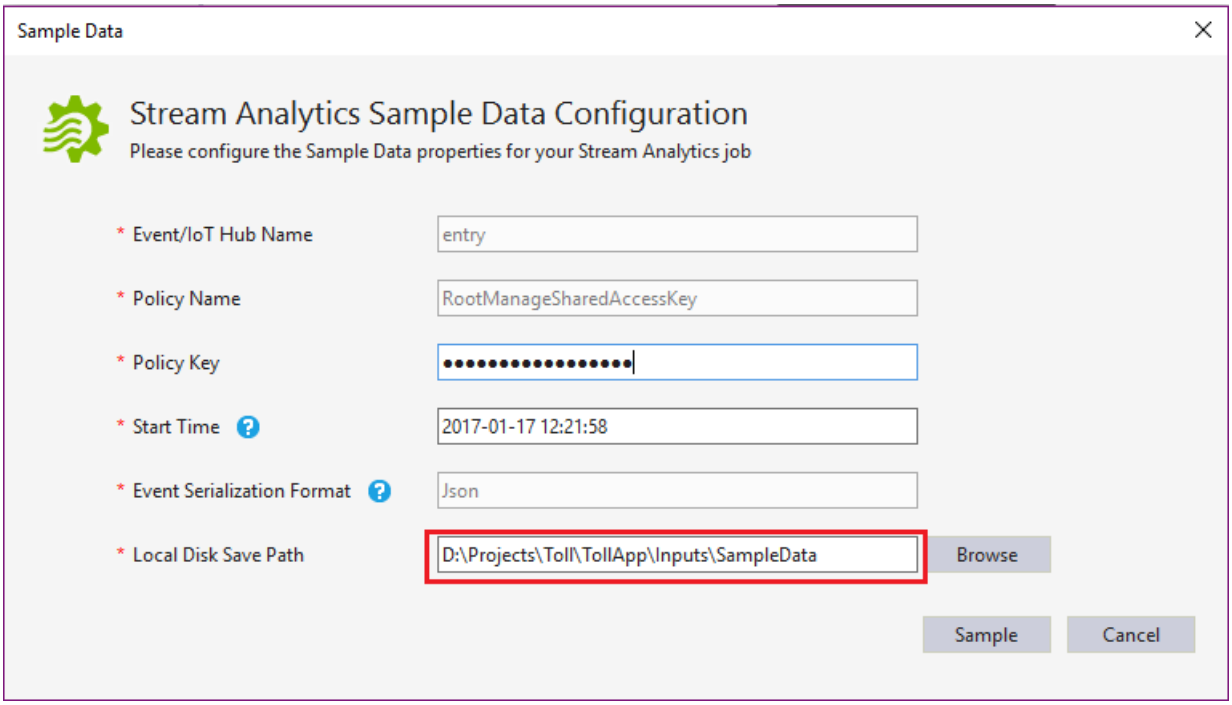


Sample input

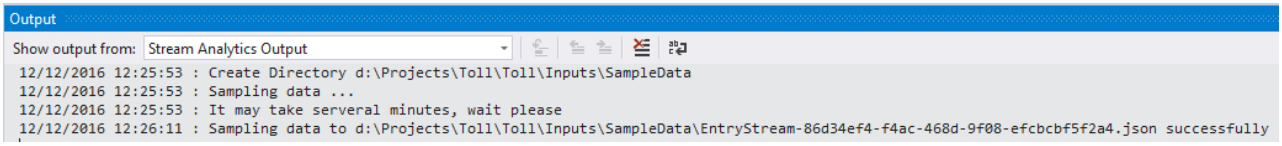
You can also sample input data from input sources to local file. Right-click the input config file and select **Sample Data**.



Please note that you can only sample Event Hub or IoT Hub for now. Other input sources are not supported. In the pop-up dialog window, please fill in the local path for saving the sample data. Click **Sample**.

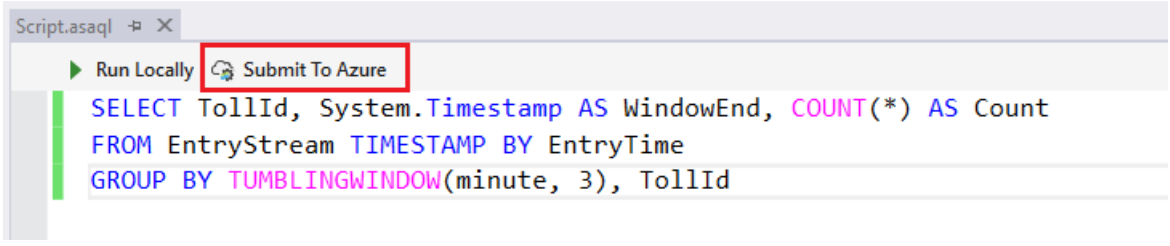


You can see the progress in the Output window.

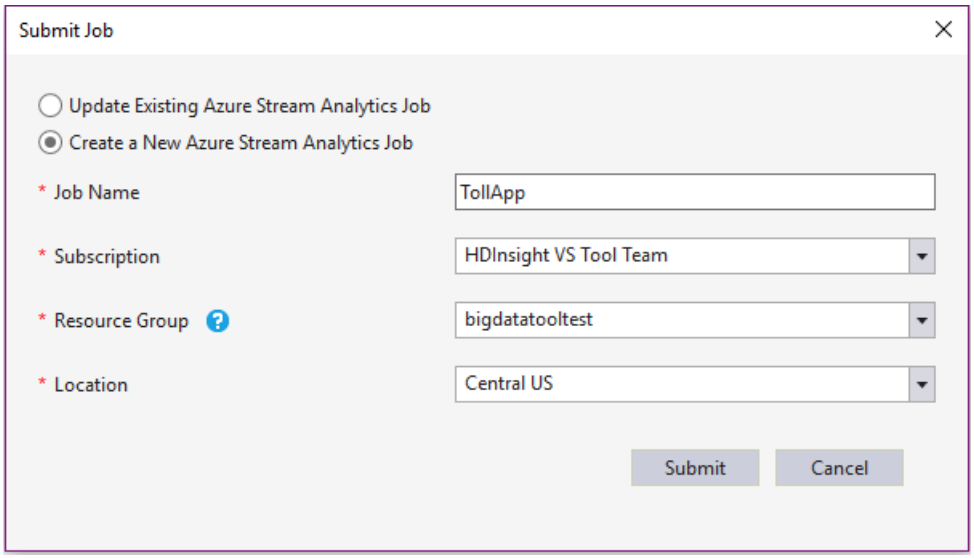


Submit Azure Stream Analytics query to Azure

In **Query Editor**, click **Submit To Azure in script editor**.

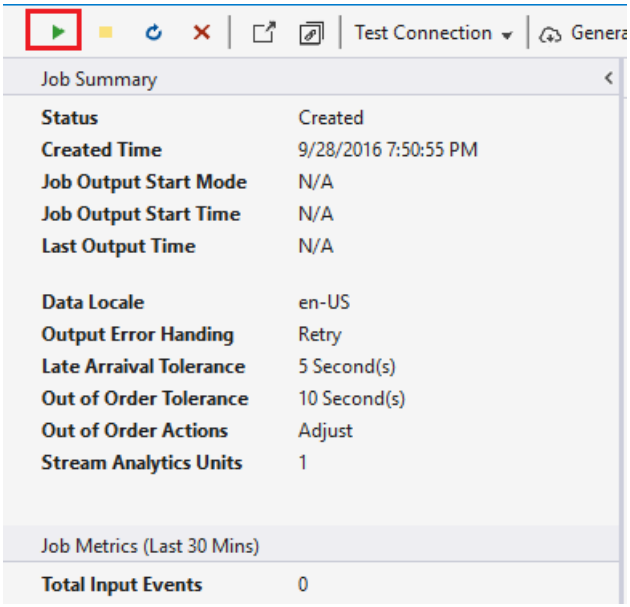


Choose Create a New Azure Stream Analytics Job. Input Job Name as below. Choose the correct Subscription. Click Submit.

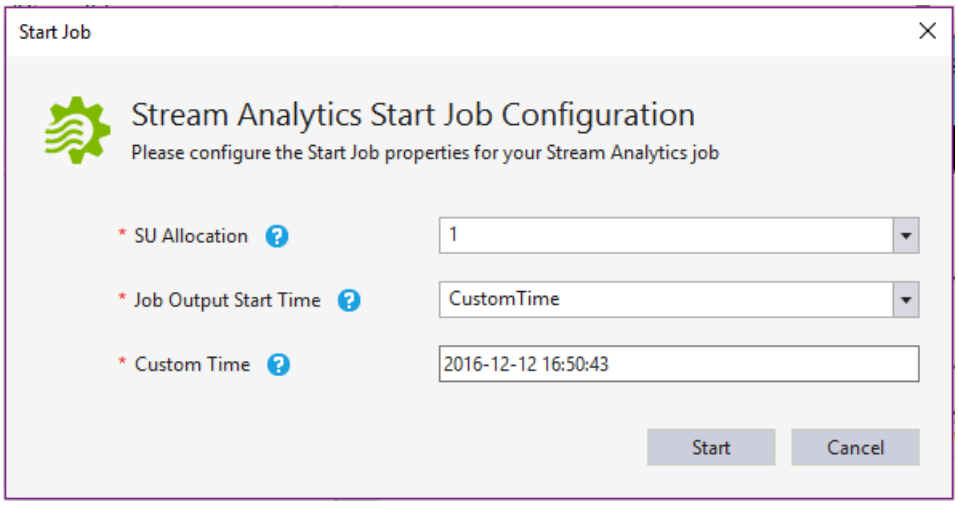


Start Job

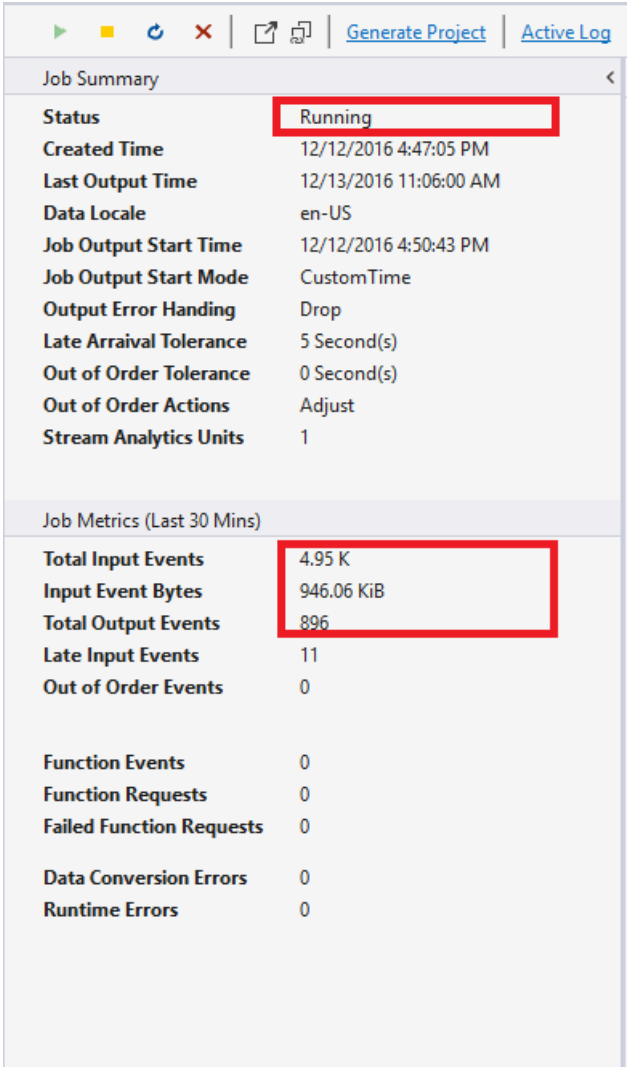
Now your job has been created and job view is automatically opened. Click the **GREEN** button to start the job.



Choose the default setting and click **Start**.

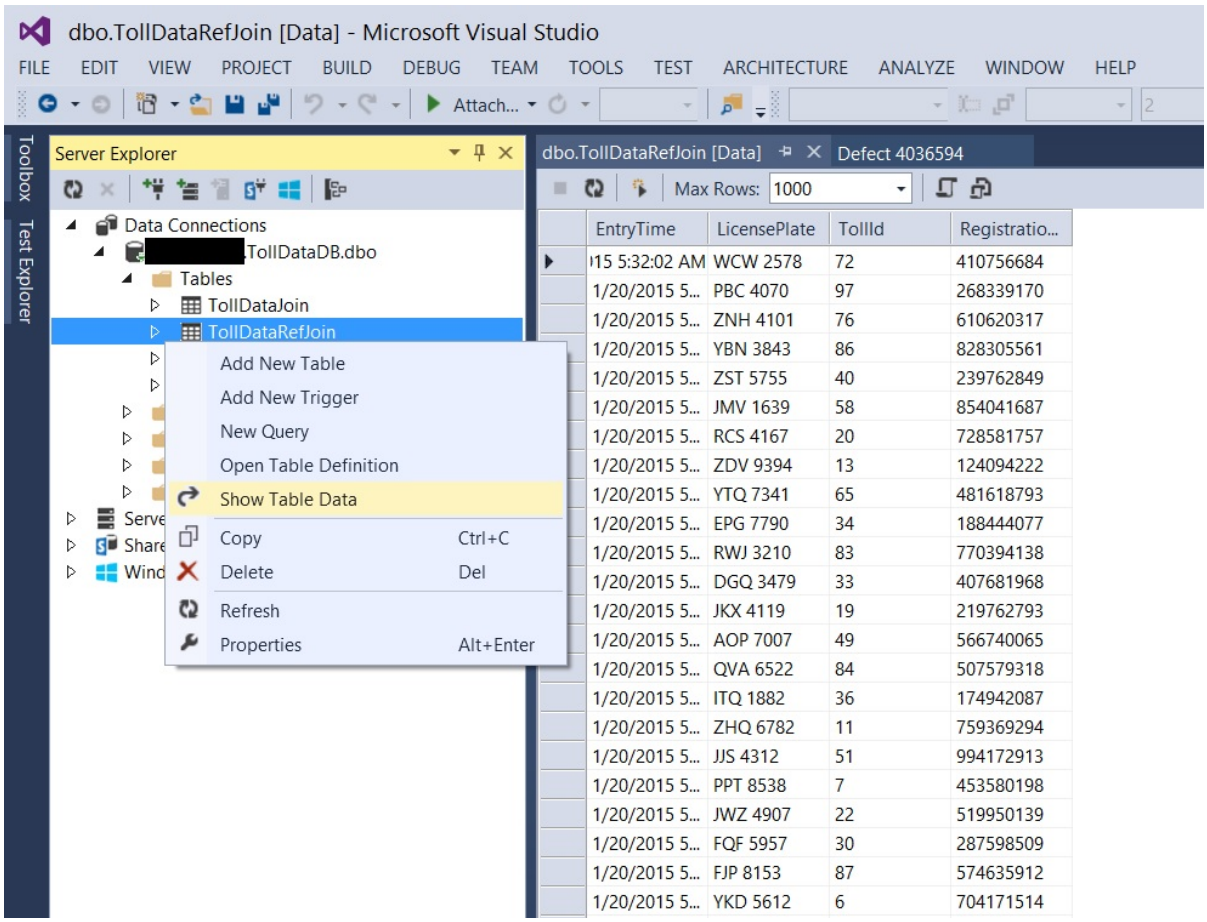


You can see the job status has changed to **Running** and there are input/output events.



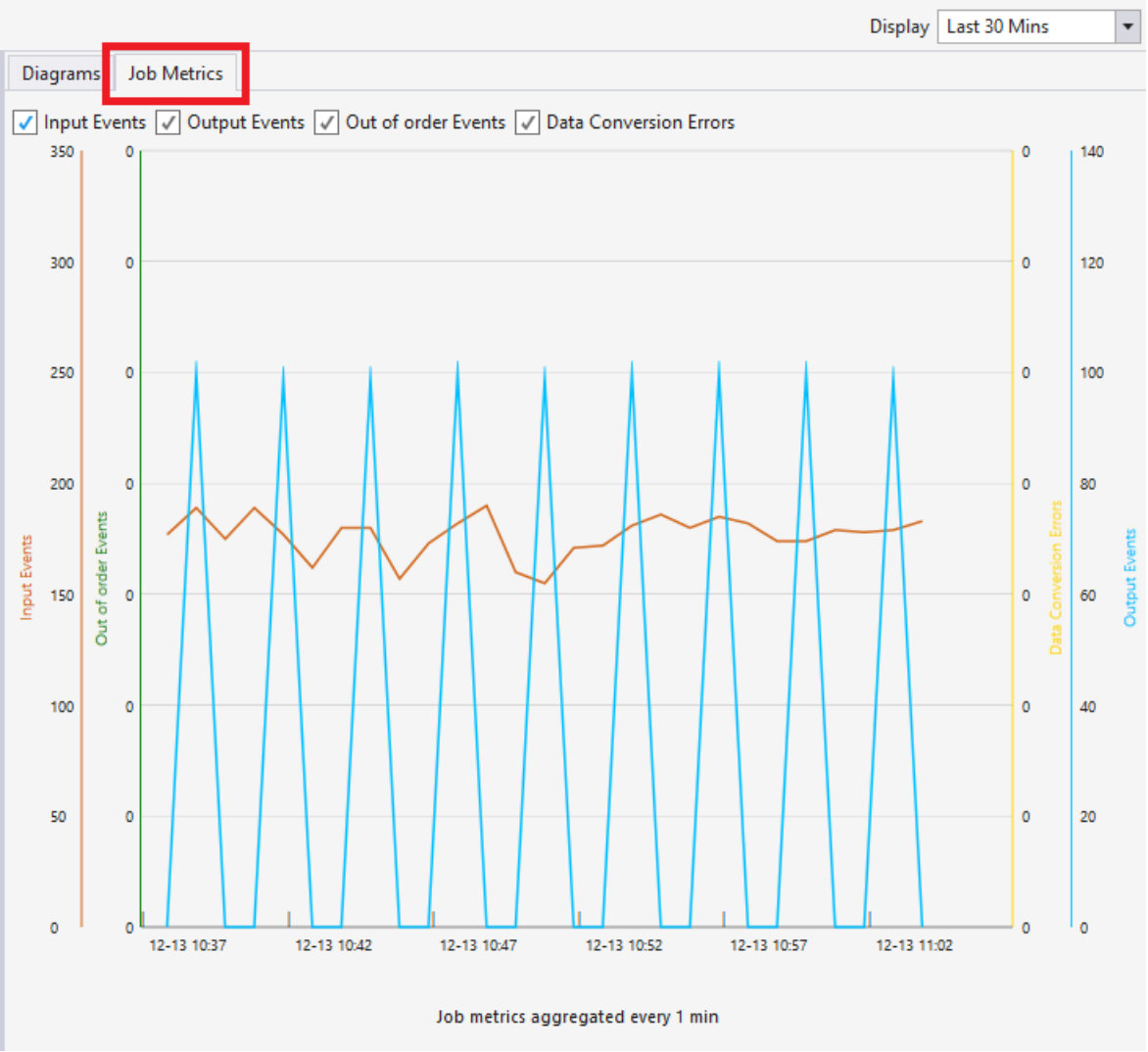
Check results in Visual Studio

1. Open Visual Studio Server Explorer, and right-click the **TollDataRefJoin** table.
2. Click **Show Table Data** to see the output of your job.



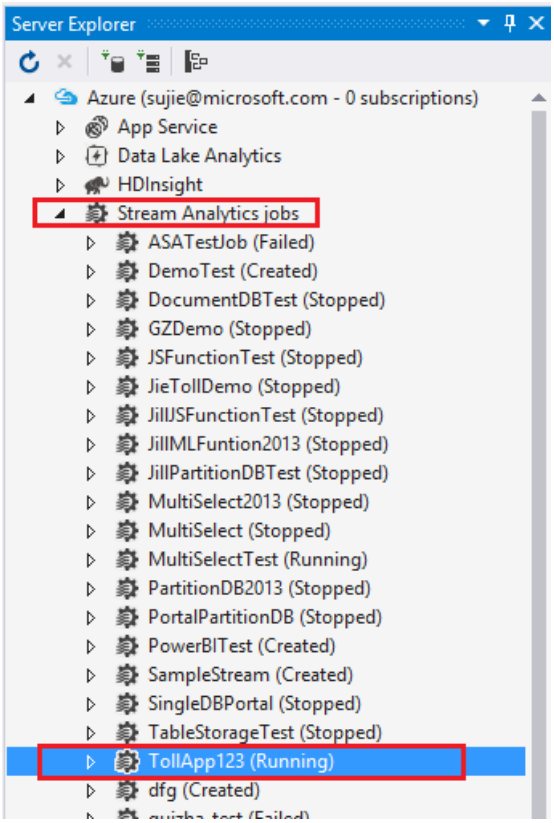
View job metrics

Some basic job statistics can be found in **Job Metrics**.



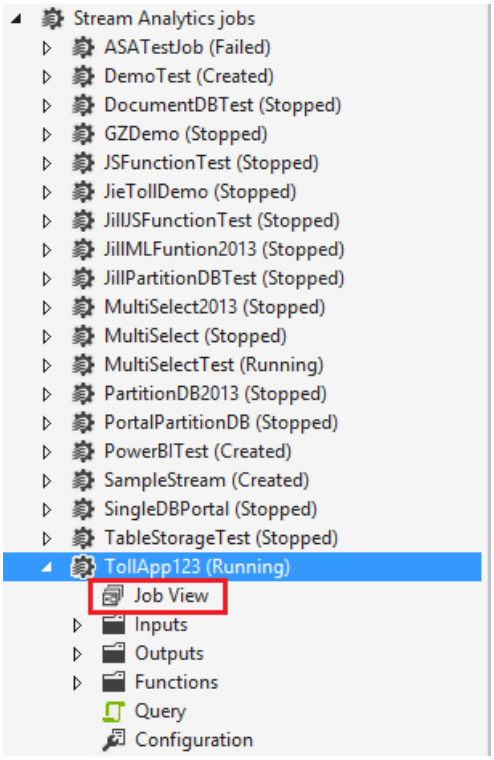
List job in Server Explorer

Click **Stream Analytics Jobs** in **Server Explorer** and click **Refresh**. You should be able to see your job appeared under **Stream Analytics Jobs**.



Open job view

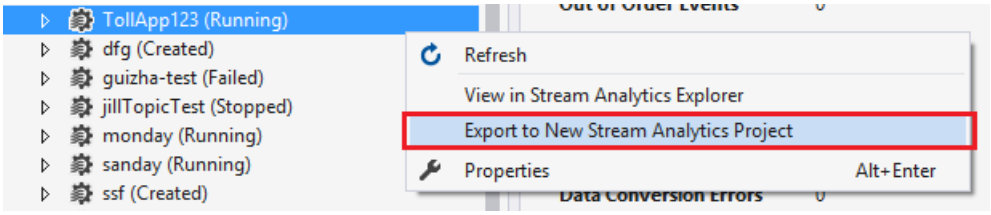
Expand your job node and double-click on the **Job View** node to open job view.



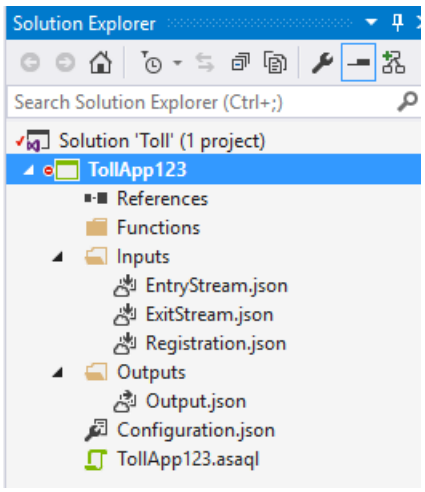
Export an existing job to a project

There are two ways you can export an existing job to a project.

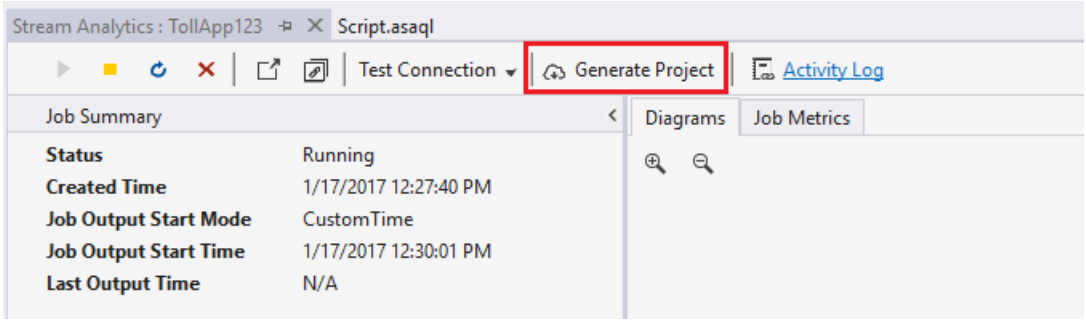
1. Right-click the job node under **Stream Analytics Jobs** node in **Server Explorer**. Click **Export to New Stream Analytics Project** from the context menu.



You will see the generated project in **Solution Explorer**.



2. In job view, click **Generate Project**.



Known Issues and Limitations

1. Local testing does not work if your query has Geo-Spatial functions.
2. No editor support for adding or changing JavaScript UDF.
3. Local testing does not support saving output in JSON format.
4. No support for Power BI output and ADLS output.

Next steps

- Introduction to Azure Stream Analytics
- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Data connection: Learn about data stream inputs from events to Stream Analytics

1/25/2017 • 9 min to read • Edit on GitHub

The data connection to Stream Analytics is a data stream of events from a data source. This is called an "input." Stream Analytics has first-class integration with Azure data stream sources Event Hub, IoT Hub, and Blob storage that can be from the same or different Azure subscription as your analytics job.

Data input types: Data stream and reference data

As data is pushed to a data source, it is consumed by the Stream Analytics job and processed in real time. Inputs are divided into two distinct types: data stream inputs and reference data inputs.

Data stream inputs

A data stream is unbounded sequence of events coming over time. Stream Analytics jobs must include at least one data stream input to be consumed and transformed by the job. Blob storage, Event Hubs, and IoT Hubs are supported as data stream input sources. Event Hubs are used to collect event streams from multiple devices and services, such as social media activity feeds, stock trade information or data from sensors. IoT Hubs are optimized to collect data from connected devices in Internet of Things (IoT) scenarios. Blob storage can be used as an input source for ingesting bulk data as a stream.

Reference data

Stream Analytics supports a second type of input known as reference data. This is auxiliary data which is either static or slowly changing over time and is typically used for performing correlation and look-ups. Azure Blob storage is currently the only supported input source for reference data. Reference data source blobs are limited to 100MB in size. To learn how to create reference data inputs, see Use Reference Data

Create a data stream input with an Event Hub

Azure Event Hubs are highly scalable publish-subscribe event ingestor. It can collect millions of events per second, so that you can process and analyze the massive amounts of data produced by your connected devices and applications. It is one of the most commonly used inputs for Stream Analytics. Event Hubs and Stream Analytics together provide customers an end to end solution for real time analytics. Event Hubs allow customers to feed events into Azure in real time, and Stream Analytics jobs can process them in real time. For example, customers can send web clicks, sensor readings, online log events to Event Hubs, and create Stream Analytics jobs to use Event Hubs as the input data streams for real time filtering, aggregating and correlation.

It is important to note that the default timestamp of events coming from Event Hubs in Stream Analytics is the timestamp that the event arrived in Event Hub which is EventEnqueuedUtcTime. To process the data as a stream using a timestamp in the event payload, the TIMESTAMP BY keyword must be used.

Consumer groups

Each Stream Analytics Event Hub input should be configured to have its own consumer group. When a job contains a self-join or multiple inputs, some input may be read by more than one reader downstream, which impacts the number of readers in a single consumer group. To avoid exceeding Event Hub limit of 5 readers per consumer group per partition, it is a best practice to designate a consumer group for each Stream Analytics job. Note that there is also a limit of 20 consumer groups per Event Hub. For details, see the Event Hubs Programming Guide.

Configure Event Hub as an input data stream

The table below explains each property in the Event Hub input tab with its description:

PROPERTY NAME	DESCRIPTION
Input Alias	A friendly name that will be used in the job query to reference this input
Service Bus Namespace	A Service Bus namespace is a container for a set of messaging entities. When you created a new Event Hub, you also created a Service Bus namespace.
Event Hub	The name of your Event Hub input.
Event Hub Policy Name	The shared access policy, which can be created on the Event Hub Configure tab. Each shared access policy will have a name, permissions that you set, and access keys.
Event Hub Policy Key	The Shared Access key used to authenticate access to the Service Bus namespace.
Event Hub Consumer Group (Optional)	The Consumer Group to ingest data from the Event Hub. If not specified, Stream Analytics jobs will use the Default Consumer Group to ingest data from the Event Hub. It is recommended to use a distinct consumer Group for each Stream Analytics job.
Event Serialization Format	To make sure your queries work the way you expect, Stream Analytics needs to know which serialization format (JSON, CSV, or Avro) you're using for incoming data streams.
Encoding	UTF-8 is the only supported encoding format at this time.

When your data is coming from an Event Hub source, you can access to few metadata fields in your Stream Analytics query. The table below lists the fields and their description.

PROPERTY	DESCRIPTION
EventProcessedUtcTime	The date and time that the event was processed by Stream Analytics.
EventEnqueuedUtcTime	The date and time that the event was received by Event Hubs.
PartitionId	The zero-based partition ID for the input adapter.

For example, you may write a query like the following:

```
SELECT
    EventProcessedUtcTime,
    EventEnqueuedUtcTime,
    PartitionId
FROM Input
```

Create an IoT Hub data stream input

Azure Iot Hub is a highly scalable publish-subscribe event ingestor optimized for IoT scenarios. It is important to note that the default timestamp of events coming from IoT Hubs in Stream Analytics is the timestamp that the

event arrived in IoT Hub which is EventEnqueuedUtcTime. To process the data as a stream using a timestamp in the event payload, the TIMESTAMP BY keyword must be used.

> **NOTE**
> Only messages sent with a DeviceClient property can be processed.

Consumer groups

Each Stream Analytics IoT Hub input should be configured to have its own consumer group. When a job contains a self-join or multiple inputs, some input may be read by more than one reader downstream, which impacts the number of readers in a single consumer group. To avoid exceeding IoT Hub limit of 5 readers per consumer group per partition, it is a best practice to designate a consumer group for each Stream Analytics job.

Configure IoT Hub as an data stream input

The table below explains each property in the IoT Hub input tab with its description:

PROPERTY NAME	DESCRIPTION
Input Alias	A friendly name that will be used in the job query to reference this input.
IoT Hub	An IoT Hub is a container for a set of messaging entities.
Endpoint	The name of your IoT Hub endpoint.
Shared Access Policy Name	The shared access policy to give access to the IoT Hub. Each shared access policy will have a name, permissions that you set, and access keys.
Shared Access Policy Key	The Shared Access key used to authenticate access to the IoT Hub.
Consumer Group (Optional)	The Consumer Group to ingest data from the IoT Hub. If not specified, Stream Analytics jobs will use the Default Consumer Group to ingest data from the IoT Hub. It is recommended to use a distinct consumer Group for each Stream Analytics job.
Event Serialization Format	To make sure your queries work the way you expect, Stream Analytics needs to know which serialization format (JSON, CSV, or Avro) you're using for incoming data streams.
Encoding	UTF-8 is the only supported encoding format at this time.

When your data is coming from an IoT Hub source, you can access to few metadata fields in your Stream Analytics query. The table below lists the fields and their description.

PROPERTY	DESCRIPTION
EventProcessedUtcTime	The date and time that the event was processed.
EventEnqueuedUtcTime	The date and time that the event was received by the IoT Hub.
PartitionId	The zero-based partition ID for the input adapter.

PROPERTY	DESCRIPTION
IoTHub.MessageId	Used to correlate two-way communication in IoT Hub.
IoTHub.CorrelationId	Used in message responses and feedback in IoT Hub.
IoTHub.ConnectionDeviceId	The authenticated id used to send this message, stamped on servicebound messages by IoT Hub.
IoTHub.ConnectionDeviceGenerationId	The generationId of the authenticated device used to send this message, Stamped on servicebound messages by IoT Hub.
IoTHub.EnqueuedTime	Time when the message was received by IoT Hub.
IoTHub.StreamId	Custom event property added by the sender device.

Create a Blob storage data stream input

For scenarios with large amounts of unstructured data to store in the cloud, Blob storage offers a cost-effective and scalable solution. Data in Blob storage is generally considered data "at rest" but it can be processed as a data stream by Stream Analytics. One common scenario for Blob storage inputs with Stream Analytics is log processing, where telemetry is captured from a system and needs to be parsed and processed to extract meaningful data.

It is important to note that the default timestamp of Blob storage events in Stream Analytics is the timestamp that the blob was last modified which *isBlobLastModifiedUtcTime*. To process the data as a stream using a timestamp in the event payload, the TIMESTAMP BY keyword must be used.

Also note that CSV formatted inputs **require** a header row to define fields for the data set. Further header row fields must all be **unique**.

> **NOTE**
> Stream Analytics does not support adding content to an existing blob. Stream Analytics will only view a blob once and any changes done after this read will not be processed. The best practice is to upload all the data once and not add any additional events to the blob store.

The table below explains each property in the Blob storage input tab with its description:

PROPERTY NAME	DESCRIPTION
Input Alias	A friendly name that will be used in the job query to reference this input.
Storage Account	The name of the storage account where your blob files are located.
Storage Account Key	The secret key associated with the storage account.
Storage Container	Containers provide a logical grouping for blobs stored in the Microsoft Azure Blob service. When you upload a blob to the Blob service, you must specify a container for that blob.

Path Prefix Pattern [optional]	The file path used to locate your blobs within the specified container. Within the path, you may choose to specify one or more instances of the following 3 variables: {date}, {time}, {partition} Example 1: cluster1/logs/{date}/{time}/{partition} Example 2: cluster1/logs/{date} Note that "*" is not an allowed value for pathprefix. Only valid Azure blob characters are allowed.
Date Format [optional]	If the date token is used in the prefix path, you can select the date format in which your files are organized. Example: YYYY/MM/DD
Time Format [optional]	If the time token is used in the prefix path, specify the time format in which your files are organized. Currently the only supported value is HH.
Event Serialization Format	To make sure your queries work the way you expect, Stream Analytics needs to know which serialization format (JSON, CSV, or Avro) you're using for incoming data streams.
Encoding	For CSV and JSON, UTF-8 is the only supported encoding format at this time.
Delimiter	Stream Analytics supports a number of common delimiters for serializing data in CSV format. Supported values are comma, semicolon, space, tab and vertical bar.

When your data is coming from a Blob storage source, you can access a few metadata fields in your Stream Analytics query. The table below lists the fields and their description.

PROPERTY	DESCRIPTION
BlobName	The name of the input blob that the event came from.
EventProcessedUtcTime	The date and time that the event was processed by Stream Analytics.
BlobLastModifiedUtcTime	The date and time that the blob was last modified.
PartitionId	The zero-based partition ID for the input adapter.

For example, you may write a query like the following:

```
SELECT
    BlobName,
    EventProcessedUtcTime,
    BlobLastModifiedUtcTime
FROM Input
```

Get help

For further assistance, try our Azure Stream Analytics forum

Next steps

You've learned about data connection options in Azure for your Stream Analytics jobs. To learn more about Stream Analytics, see:

- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Stream Analytics outputs: Options for storage, analysis

1/25/2017 • 16 min to read • Edit on GitHub

When authoring a Stream Analytics job, consider how the resulting data will be consumed. How will you view the results of the Stream Analytics job and where will you store it?

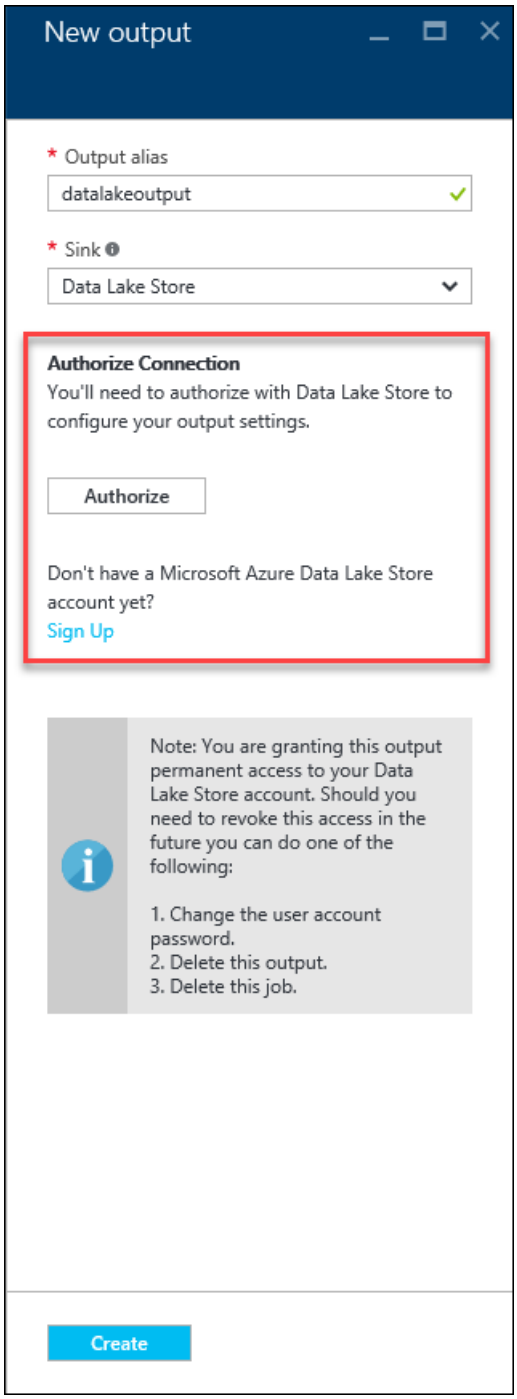
In order to enable a variety of application patterns, Azure Stream Analytics has different options for storing output and viewing analysis results. This makes it easy to view job output and gives you flexibility in the consumption and storage of the job output for data warehousing and other purposes. Any output configured in the job must exist before the job is started and events start flowing. For example, if you use Blob storage as an output, the job will not create a storage account automatically. It needs to be created by the user before the ASA job is started.

Azure Data Lake Store

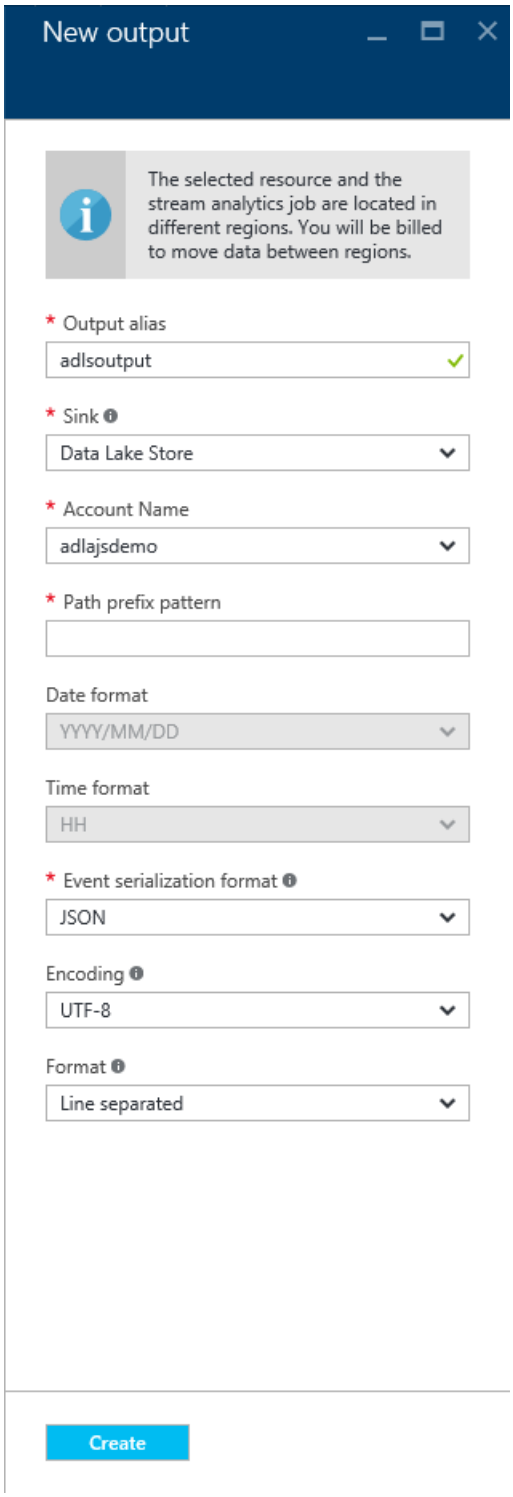
Stream Analytics supports Azure Data Lake Store. This storage enables you to store data of any size, type and ingestion speed for operational and exploratory analytics. Further, Stream Analytics needs to be authorized to access the Data Lake Store. Details on authorization and how to sign up for the Data Lake Store (if needed) are discussed in the Data Lake output article.

Authorize an Azure Data Lake Store

When Data Lake Storage is selected as an output in the Azure Management portal, you will be prompted to authorize a connection to an existing Data Lake Store.



Then fill out the properties for the Data Lake Store output as seen below:



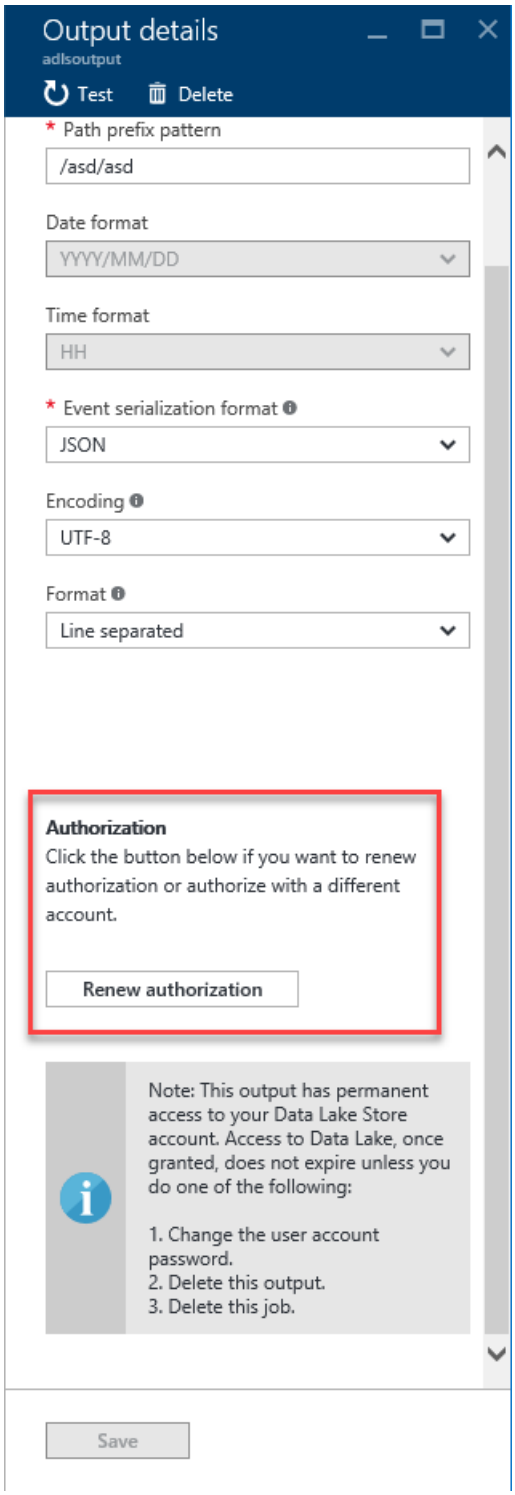
The table below lists the property names and their description needed for creating a Data Lake Store output.

PROPERTY NAME	DESCRIPTION
Output Alias	This is a friendly name used in queries to direct the query output to this Data Lake Store.
Account Name	The name of the Data Lake Storage account where you are sending your output. You will be presented with a drop down list of Data Lake Store accounts to which the user logged in to the portal has access to.

Path Prefix Pattern [*optional*]	The file path used to write your files within the specified Data Lake Store Account. {date}, {time} Example 1: folder1/logs/{date}/{time} Example 2: folder1/logs/{date}
Date Format [*optional*]	If the date token is used in the prefix path, you can select the date format in which your files are organized. Example: YYYY/MM/DD
Time Format [*optional*]	If the time token is used in the prefix path, specify the time format in which your files are organized. Currently the only supported value is HH.
Event Serialization Format	Serialization format for output data. JSON, CSV, and Avro are supported.
Encoding	If CSV or JSON format, an encoding must be specified. UTF-8 is the only supported encoding format at this time.
Delimiter	Only applicable for CSV serialization. Stream Analytics supports a number of common delimiters for serializing CSV data. Supported values are comma, semicolon, space, tab and vertical bar.
Format	Only applicable for JSON serialization. Line separated specifies that the output will be formatted by having each JSON object separated by a new line. Array specifies that the output will be formatted as an array of JSON objects.

Renew Data Lake Store Authorization

You will need to re-authenticate your Data Lake Store account if its password has changed since your job was created or last authenticated.



SQL Database

Azure SQL Database can be used as an output for data that is relational in nature or for applications that depend on content being hosted in a relational database. Stream Analytics jobs will write to an existing table in an Azure SQL Database. Note that the table schema must exactly match the fields and their types being output from your job. An Azure SQL Data Warehouse can also be specified as an output via the SQL Database output option as well (this is a preview feature). The table below lists the property names and their description for creating a SQL Database output.

PROPERTY NAME	DESCRIPTION
Output Alias	This is a friendly name used in queries to direct the query output to this database.

PROPERTY NAME	DESCRIPTION
Database	The name of the database where you are sending your output
Server Name	The SQL Database server name
Username	The Username which has access to write to the database
Password	The password to connect to the database
Table	The table name where the output will be written. The table name is case sensitive and the schema of this table should match exactly to the number of fields and their types being generated by your job output.

> **NOTE**
> Currently the Azure SQL Database offering is supported for a job output in Stream Analytics. However, an Azure Virtual Machine running SQL Server with a database attached is not supported. This is subject to change in future releases.

Blob storage

Blob storage offers a cost-effective and scalable solution for storing large amounts of unstructured data in the cloud. For an introduction on Azure Blob storage and its usage, see the documentation at How to use Blobs.

The table below lists the property names and their description for creating a blob output.

PROPERTY NAME	DESCRIPTION
Output Alias	This is a friendly name used in queries to direct the query output to this blob storage.
Storage Account	The name of the storage account where you are sending your output.
Storage Account Key	The secret key associated with the storage account.
Storage Container	Containers provide a logical grouping for blobs stored in the Microsoft Azure Blob service. When you upload a blob to the Blob service, you must specify a container for that blob.
Path Prefix Pattern [optional]	The file path used to write your blobs within the specified container. Within the path, you may choose to use one or more instances of the following 2 variables to specify the frequency that blobs are written: {date}, {time} Example 1: cluster1/logs/{date}/{time} Example 2: cluster1/logs/{date}
Date Format [optional]	If the date token is used in the prefix path, you can select the date format in which your files are organized. Example: YYYY/MM/DD

Time Format [optional]	If the time token is used in the prefix path, specify the time format in which your files are organized. Currently the only supported value is HH.
Event Serialization Format	Serialization format for output data. JSON, CSV, and Avro are supported.
Encoding	If CSV or JSON format, an encoding must be specified. UTF-8 is the only supported encoding format at this time.
Delimiter	Only applicable for CSV serialization. Stream Analytics supports a number of common delimiters for serializing CSV data. Supported values are comma, semicolon, space, tab and vertical bar.
Format	Only applicable for JSON serialization. Line separated specifies that the output will be formatted by having each JSON object separated by a new line. Array specifies that the output will be formatted as an array of JSON objects.

Event Hub

Event Hubs is a highly scalable publish-subscribe event ingestor. It can collect millions of events per second. One use of an Event Hub as output is when the output of a Stream Analytics job will be the input of another streaming job.

There are a few parameters that are needed to configure Event Hub data streams as an output.

PROPERTY NAME	DESCRIPTION
Output Alias	This is a friendly name used in queries to direct the query output to this Event Hub.
Service Bus Namespace	A Service Bus namespace is a container for a set of messaging entities. When you created a new Event Hub, you also created a Service Bus namespace
Event Hub	The name of your Event Hub output
Event Hub Policy Name	The shared access policy, which can be created on the Event Hub Configure tab. Each shared access policy will have a name, permissions that you set, and access keys
Event Hub Policy Key	The Shared Access key used to authenticate access to the Service Bus namespace
Partition Key Column [optional]	This column contains the partition key for Event Hub output.
Event Serialization Format	Serialization format for output data. JSON, CSV, and Avro are supported.
Encoding	For CSV and JSON, UTF-8 is the only supported encoding format at this time

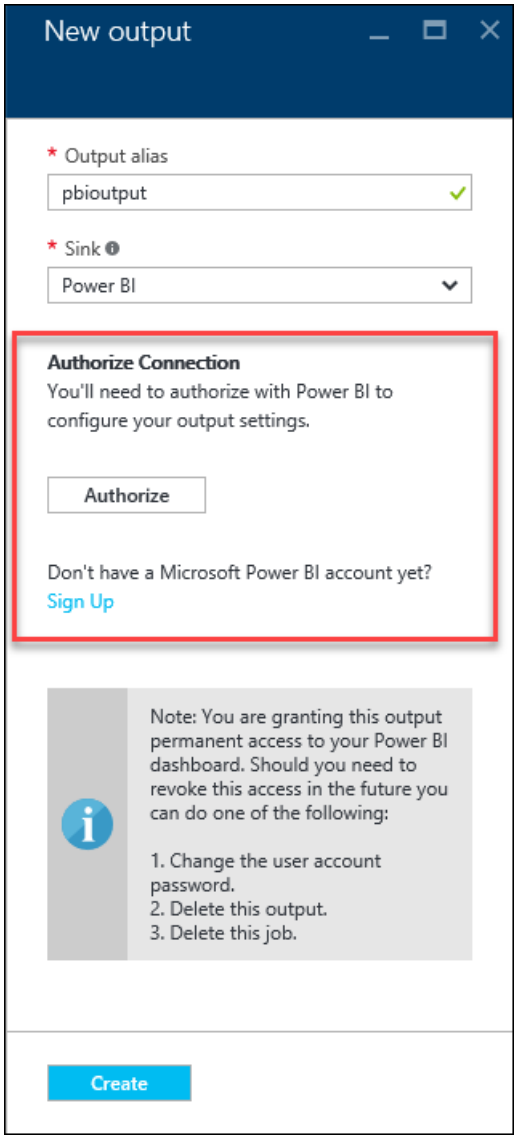
PROPERTY NAME	DESCRIPTION
Delimiter	Only applicable for CSV serialization. Stream Analytics supports a number of common delimiters for serializing data in CSV format. Supported values are comma, semicolon, space, tab and vertical bar.
Format	Only applicable for JSON type. Line separated specifies that the output will be formatted by having each JSON object separated by a new line. Array specifies that the output will be formatted as an array of JSON objects.

Power BI

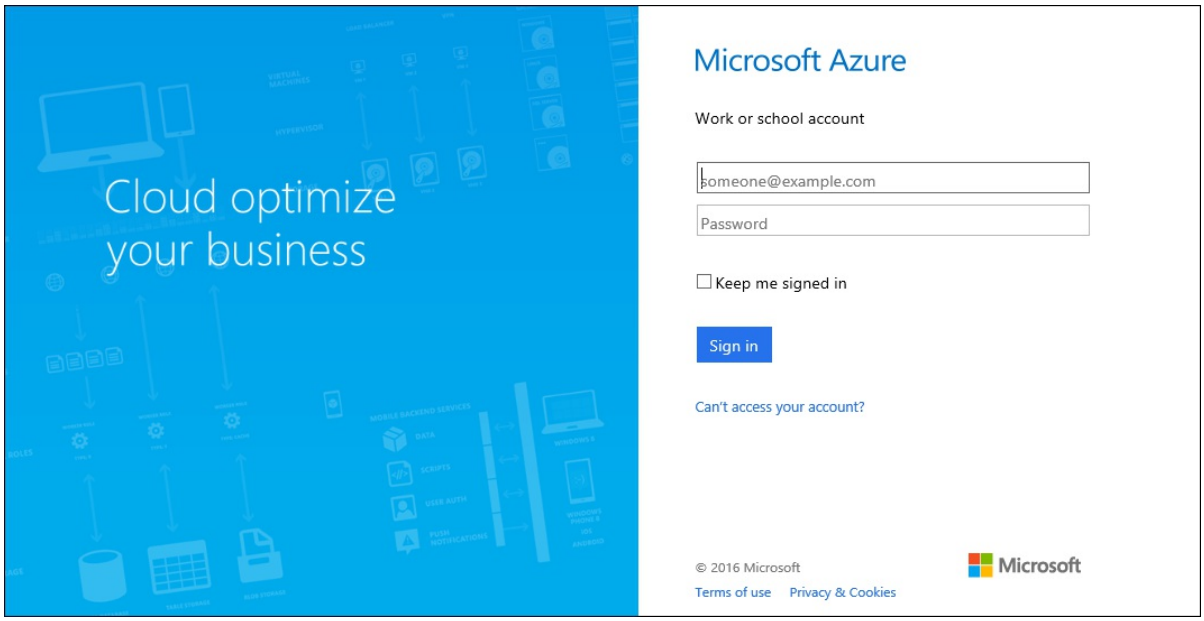
Power BI can be used as an output for a Stream Analytics job to provide for a rich visualization experience of analysis results. This capability can be used for operational dashboards, report generation and metric driven reporting.

Authorize a Power BI account

1. When Power BI is selected as an output in the Azure Management portal, you will be prompted to authorize an existing Power BI User or to create a new Power BI account.



2. Create a new account if you don't yet have one, then click Authorize Now. A screen like the following is presented.



3. In this step, provide the work or school account for authorizing the Power BI output. If you are not already signed up for Power BI, choose Sign up now. The work or school account you use for Power BI could be different from the Azure subscription account which you are currently logged in with.

Configure the Power BI output properties

Once you have the Power BI account authenticated, you can configure the properties for your Power BI output. The table below is the list of property names and their description to configure your Power BI output.

PROPERTY NAME	DESCRIPTION
Output Alias	This is a friendly name used in queries to direct the query output to this PowerBI output.
Group Workspace	To enable sharing data with other Power BI users you can select groups inside your Power BI account or choose "My Workspace" if you do not want to write to a group. Updating an existing group requires renewing the Power BI authentication.
Dataset Name	Provide a dataset name that it is desired for the Power BI output to use
Table Name	Provide a table name under the dataset of the Power BI output. Currently, Power BI output from Stream Analytics jobs can only have one table in a dataset

For a walk-through of configuring a Power BI output and dashboard, please see the Azure Stream Analytics & Power BI article.

> **NOTE**
> Do not explicitly create the dataset and table in the Power BI dashboard. The dataset and table will be automatically populated when the job is started and the job starts pumping output into Power BI. Note that if the job query doesn't generate any results, the dataset and table will not be created. Also be aware that if Power BI already had a dataset and table with the same name as the one provided in this Stream Analytics job, the existing data will be overwritten.

Schema Creation

Azure Stream Analytics creates a Power BI dataset and table on behalf of the user if one does not already exist. In

all other cases, the table is updated with new values.Currently, there is a the limitation that only one table can exist within a dataset.

Data type conversion from ASA to Power BI

Azure Stream Analytics updates the data model dynamically at runtime if the output schema changes. Column name changes, column type changes, and the addition or removal of columns are all tracked.

This table covers the data type conversions from Stream Analytics data types to Power BIs Entity Data Model (EDM) types if a POWER BI dataset and table do not exist.

FROM STREAM ANALYTICS	TO POWER BI
bigint	Int64
nvarchar(max)	String
datetime	Datetime
float	Double
Record array	String type, Constant value "IRecord" or "IArray"

Schema Update

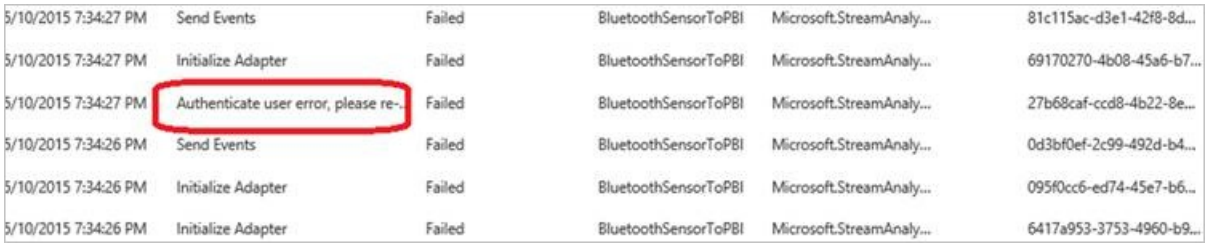
Stream Analytics infers the data model schema based on the first set of events in the output. Later, if necessary, the data model schema is updated to accommodate incoming events that may not fit into the original schema.

The `SELECT *` query should be avoided to prevent dynamic schema update across rows. In addition to potential performance implications, it could also result in indeterminacy of the time taken for the results. The exact fields that need to be shown on Power BI dashboard should be selected. Additionally, the data values should be compliant with the chosen data type.

PREVIOUS/CURRENT	INT64	STRING	DATETIME	DOUBLE
Int64	Int64	String	String	Double
Double	Double	String	String	Double
String	String	String	String	
Datetime	String	String	Datetime	String

Renew Power BI Authorization

You will need to re-authenticate your Power BI account if its password has changed since your job was created or last authenticated. If Multi-Factor Authentication (MFA) is configured on your Azure Active Directory (AAD) tenant you will also need to renew Power BI authorization every 2 weeks. A symptom of this issue is no job output and an "Authenticate user error" in the Operation Logs:



To resolve this issue, stop your running job and go to your Power BI output. Click the "Renew authorization" link, and restart your job from the Last Stopped Time to avoid data loss.

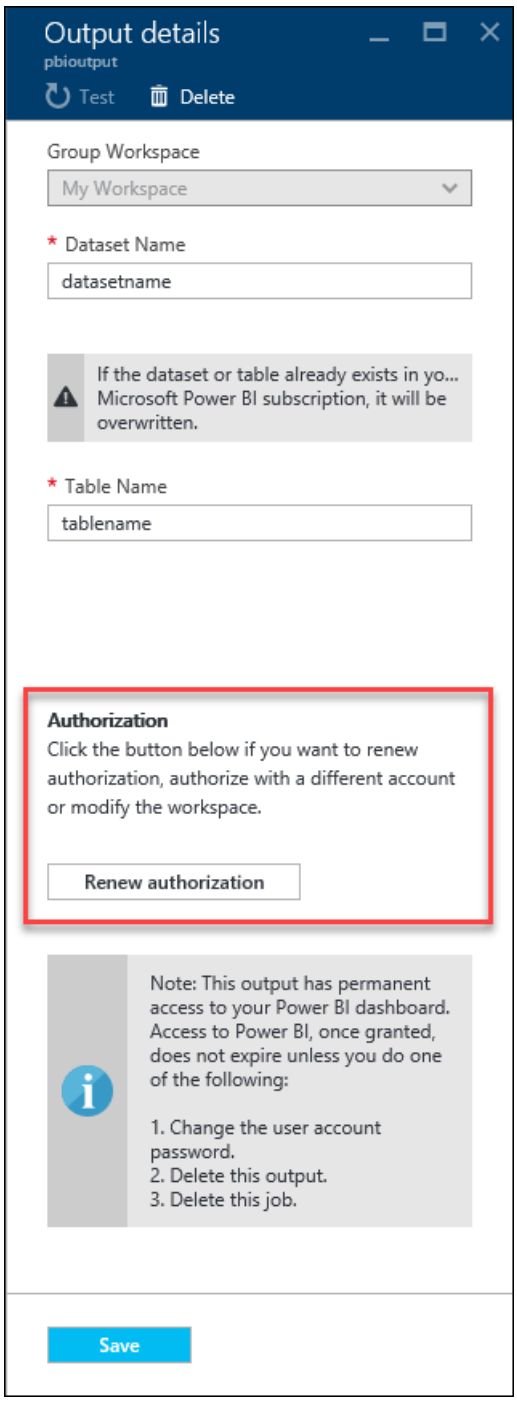


Table Storage

Azure Table storage offers highly available, massively scalable storage, so that an application can automatically scale to meet user demand. Table storage is Microsoft's NoSQL key/attribute store which one can leverage for structured data with less constraints on the schema. Azure Table storage can be used to store data for persistence and efficient retrieval.

The table below lists the property names and their description for creating a table output.

PROPERTY NAME	DESCRIPTION
Output Alias	This is a friendly name used in queries to direct the query output to this table storage.

PROPERTY NAME	DESCRIPTION
Storage Account	The name of the storage account where you are sending your output.
Storage Account Key	The access key associated with the storage account.
Table Name	The name of the table. The table will get created if it does not exist.
Partition Key	The name of the output column containing the partition key. The partition key is a unique identifier for the partition within a given table that forms the first part of an entity's primary key. It is a string value that may be up to 1 KB in size.
Row Key	The name of the output column containing the row key. The row key is a unique identifier for an entity within a given partition. It forms the second part of an entity's primary key. The row key is a string value that may be up to 1 KB in size.
Batch Size	The number of records for a batch operation. Typically the default is sufficient for most jobs, refer to the Table Batch Operation spec for more details on modifying this setting.

Service Bus Queues

Service Bus Queues offer a First In, First Out (FIFO) message delivery to one or more competing consumers. Typically, messages are expected to be received and processed by the receivers in the temporal order in which they were added to the queue, and each message is received and processed by only one message consumer.

The table below lists the property names and their description for creating a Queue output.

PROPERTY NAME	DESCRIPTION
Output Alias	This is a friendly name used in queries to direct the query output to this Service Bus Queue.
Service Bus Namespace	A Service Bus namespace is a container for a set of messaging entities.
Queue Name	The name of the Service Bus Queue.
Queue Policy Name	When you create a Queue, you can also create shared access policies on the Queue Configure tab. Each shared access policy will have a name, permissions that you set, and access keys.
Queue Policy Key	The Shared Access key used to authenticate access to the Service Bus namespace
Event Serialization Format	Serialization format for output data. JSON, CSV, and Avro are supported.
Encoding	For CSV and JSON, UTF-8 is the only supported encoding format at this time

PROPERTY NAME	DESCRIPTION
Delimiter	Only applicable for CSV serialization. Stream Analytics supports a number of common delimiters for serializing data in CSV format. Supported values are comma, semicolon, space, tab and vertical bar.
Format	Only applicable for JSON type. Line separated specifies that the output will be formatted by having each JSON object separated by a new line. Array specifies that the output will be formatted as an array of JSON objects.

Service Bus Topics

While Service Bus Queues provide a one to one communication method from sender to receiver, Service Bus Topics provide a one-to-many form of communication.

The table below lists the property names and their description for creating a table output.

PROPERTY NAME	DESCRIPTION
Output Alias	This is a friendly name used in queries to direct the query output to this Service Bus Topic.
Service Bus Namespace	A Service Bus namespace is a container for a set of messaging entities. When you created a new Event Hub, you also created a Service Bus namespace
Topic Name	Topics are messaging entities, similar to event hubs and queues. They're designed to collect event streams from a number of different devices and services. When a topic is created, it is also given a specific name. The messages sent to a Topic will not be available unless a subscription is created, so ensure there are one or more subscriptions under the topic
Topic Policy Name	When you create a Topic, you can also create shared access policies on the Topic Configure tab. Each shared access policy will have a name, permissions that you set, and access keys
Topic Policy Key	The Shared Access key used to authenticate access to the Service Bus namespace
Event Serialization Format	Serialization format for output data. JSON, CSV, and Avro are supported.
Encoding	If CSV or JSON format, an encoding must be specified. UTF-8 is the only supported encoding format at this time
Delimiter	Only applicable for CSV serialization. Stream Analytics supports a number of common delimiters for serializing data in CSV format. Supported values are comma, semicolon, space, tab and vertical bar.

DocumentDB

Azure DocumentDB is a fully-managed NoSQL document database service that offers query and transactions over

schema-free data, predictable and reliable performance, and rapid development.

The below list details the property names and their description for creating a DocumentDB output.

- **Output Alias** – An alias to refer this output in your ASA query
- **Account Name** – The name or endpoint URI of the DocumentDB account.
- **Account Key** – The shared access key for the DocumentDB account.
- **Database** – The DocumentDB database name.
- **Collection Name Pattern** – The collection name or their pattern for the collections to be used. The collection name format can be constructed using the optional {partition} token, where partitions start from 0. Following are sample valid inputs:
  1) MyCollection – One collection named "MyCollection" must exist.
  2) MyCollection{partition} – Such collections must exist– "MyCollection0", "MyCollection1", "MyCollection2" and so on.
- **Partition Key** – Optional. This is only needed if you are using a {parition} token in your collection name pattern. The name of the field in output events used to specify the key for partitioning output across collections. For single collection output, any arbitrary output column can be used e.g. PartitionId.
- **Document ID** – Optional. The name of the field in output events used to specify the primary key on which insert or update operations are based.

Get help

For further assistance, try our Azure Stream Analytics forum

Next steps

You've been introduced to Stream Analytics, a managed service for streaming analytics on data from the Internet of Things. To learn more about this service, see:

- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Using reference data or lookup tables in a Stream Analytics input stream

2/7/2017 • 5 min to read • Edit on GitHub

Reference data (also known as a lookup table) is a finite data set that is static or slowing changing in nature, used to perform a lookup or to correlate with your data stream. To make use of reference data in your Azure Stream Analytics job, you will generally use a Reference Data Join in your Query. Stream Analytics uses Azure Blob storage as the storage layer for Reference Data, and with Azure Data Factory reference data can be transformed and/or copied to Azure Blob storage, for use as Reference Data, from any number of cloud-based and on-premises data stores. Reference data is modeled as a sequence of blobs (defined in the input configuration) in ascending order of the date/time specified in the blob name. It **only** supports adding to the end of the sequence by using a date/time **greater** than the one specified by the last blob in the sequence.

Stream Analytics has a **limit of 100 MB per blob** but jobs can process multiple reference blobs by using the **path pattern** property.

Configuring reference data

To configure your reference data, you first need to create an input that is of type **Reference Data**. The table below explains each property that you will need to provide while creating the reference data input with its description:

Property Name	Description
Input Alias	A friendly name that will be used in the job query to reference this input.
Storage Account	The name of the storage account where your blobs are located. If it's in the same subscription as your Stream Analytics Job, you can select it from the drop-down.
Storage Account Key	The secret key associated with the storage account. This gets automatically populated if the storage account is in the same subscription as your Stream Analytics job.
Storage Container	Containers provide a logical grouping for blobs stored in the Microsoft Azure Blob service. When you upload a blob to the Blob service, you must specify a container for that blob.
Path Pattern	The path used to locate your blobs within the specified container. Within the path, you may choose to specify one or more instances of the following 2 variables: {date}, {time} Example 1: products/{date}/{time}/product-list.csv Example 2: products/{date}/product-list.csv
Date Format [optional]	If you have used {date} within the Path Pattern that you specified, then you can select the date format in which your blobs are organized from the drop-down of supported formats. Example: YYYY/MM/DD, MM/DD/YYYY, etc.

Time Format [optional]	If you have used {time} within the Path Pattern that you specified, then you can select the time format in which your blobs are organized from the drop-down of supported formats. Example: HH, HH/mm, or HH-mm
Event Serialization Format	To make sure your queries work the way you expect, Stream Analytics needs to know which serialization format you're using for incoming data streams. For Reference Data, the supported formats are CSV and JSON.
Encoding	UTF-8 is the only supported encoding format at this time

Generating reference data on a schedule

If your reference data is a slowly changing data set, then support for refreshing reference data is enabled by specifying a path pattern in the input configuration using the {date} and {time} substitution tokens. Stream Analytics picks up the updated reference data definitions based on this path pattern. For example, a pattern of `sample/{date}/{time}/products.csv` with a date format of **"YYYY-MM-DD"** and a time format of **"HH-mm"** instructs Stream Analytics to pick up the updated blob `sample/2015-04-16/17-30/products.csv` at 5:30 PM on April 16th, 2015 UTC time zone.

> **NOTE**
>
> Currently Stream Analytics jobs look for the blob refresh only when the machine time advances to the time encoded in the blob name. For example, the job will look for `sample/2015-04-16/17-30/products.csv` as soon as possible but no earlier than 5:30 PM on April 16th, 2015 UTC time zone. It will *never* look for a blob with an encoded time earlier than the last one that is discovered.
>
> E.g. once the job finds the blob `sample/2015-04-16/17-30/products.csv` it will ignore any files with an encoded date earlier than 5:30 PM April 16th, 2015 so if a late arriving `sample/2015-04-16/17-25/products.csv` blob gets created in the same container the job will not use it.
>
> Likewise if `sample/2015-04-16/17-30/products.csv` is only produced at 10:03 PM April 16th, 2015 but no blob with an earlier date is present in the container, the job will use this file starting at 10:03 PM April 16th, 2015 and use the previous reference data until then.
>
> An exception to this is when the job needs to re-process data back in time or when the job is first started. At start time the job is looking for the most recent blob produced before the job start time specified. This is done to ensure that there is a **non-empty** reference data set when the job starts. If one cannot be found, the job displays the following diagnostic: `Initializing input without a valid reference data blob for UTC time <start time>`.

Azure Data Factory can be used to orchestrate the task of creating the updated blobs required by Stream Analytics to update reference data definitions. Data Factory is a cloud-based data integration service that orchestrates and automates the movement and transformation of data. Data Factory supports connecting to a large number of cloud based and on-premises data stores and moving data easily on a regular schedule that you specify. For more information and step by step guidance on how to set up a Data Factory pipeline to generate reference data for Stream Analytics which refreshes on a pre-defined schedule, check out this GitHub sample.

Tips on refreshing your reference data

1. Overwriting reference data blobs will not cause Stream Analytics to reload the blob and in some cases it can cause the job to fail. The recommended way to change reference data is to add a new blob using the same container and path pattern defined in the job input and use a date/time **greater** than the one specified by the last blob in the sequence.

2. Reference data blobs are **not** ordered by the blob's "Last Modified" time but only by the time and date specified in the blob name using the {date} and {time} substitutions.
3. On a few occasions, a job must go back in time, therefore reference data blobs must not be altered or deleted.

Get help

For further assistance, try our Azure Stream Analytics forum

Next steps

You've been introduced to Stream Analytics, a managed service for streaming analytics on data from the Internet of Things. To learn more about this service, see:

- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

How to store data from Azure Stream Analytics in an Azure Redis Cache using Azure Functions

1/23/2017 • 7 min to read • Edit on GitHub

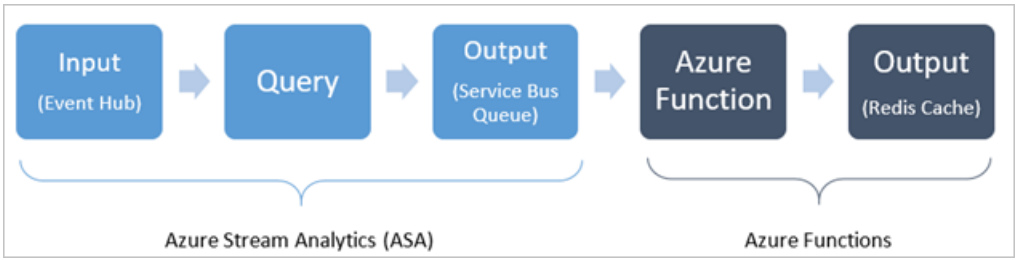
Azure Stream Analytics lets you rapidly develop and deploy low-cost solutions to gain real-time insights from devices, sensors, infrastructure, and applications, or any stream of data. It enables various use cases such as real-time management and monitoring, command and control, fraud detection, connected cars and many more. In many such scenarios, you may want to store data outputted by Azure Stream Analytics into a distributed data store such as an Azure Redis cache.

Suppose you are part of a telecommunications company. You are trying to detect SIM fraud where multiple calls coming from the same identity, at the same time, but in different geographically locations. You are tasked with storing all the potential fraudulent phone calls in an Azure Redis cache. In this blog, we provide guidance on how you can easily complete your task.

Prerequisites

Complete the Real-time Fraud Detection walk-through for ASA

Architecture Overview



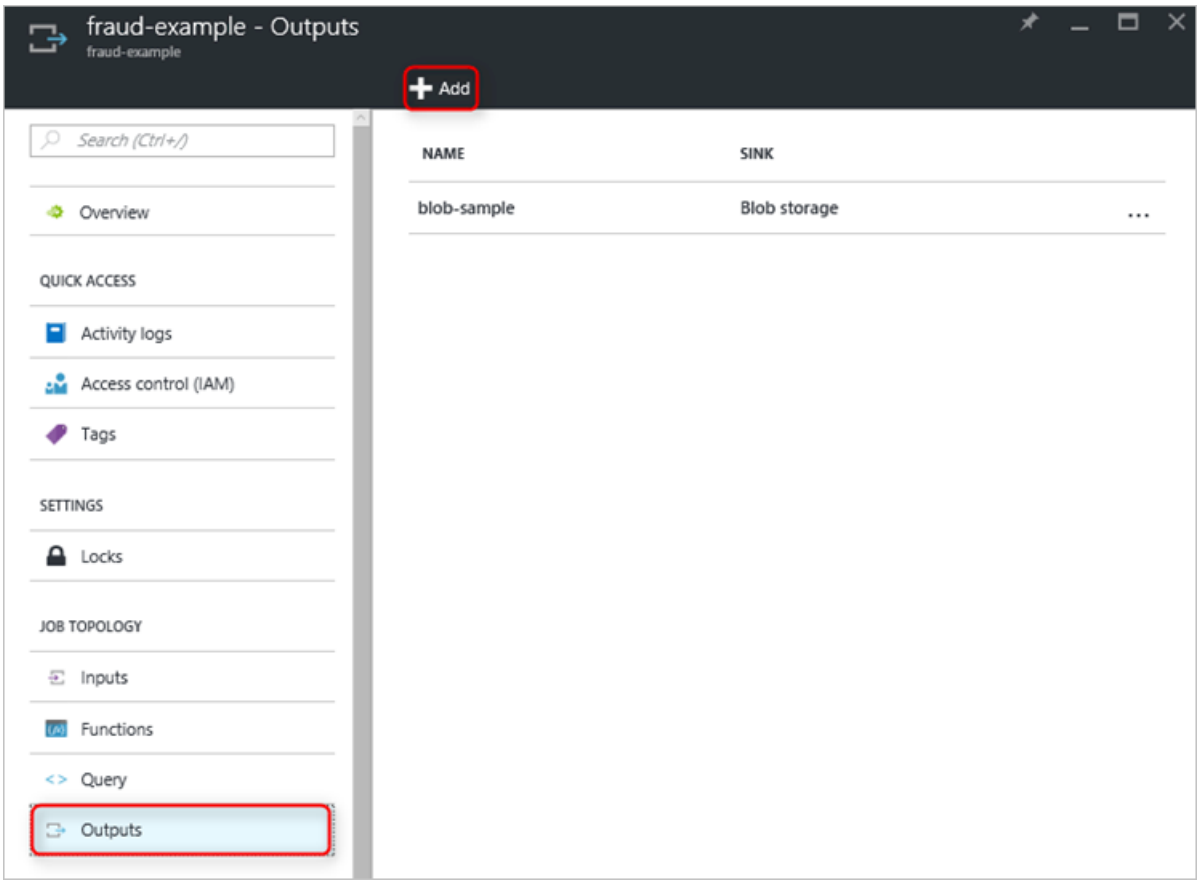
As shown in the preceding figure, Stream Analytics allows streaming input data to be queried and sent to an output. Based on the output, Azure Functions can then trigger some type of event.

In this blog, we focus on the Azure Functions part of this pipeline, or more specifically the triggering of an event that stores fraudulent data into the cache. After completing the Real-time Fraud Detection tutorial, you have an input (an event hub), a query, and an output (blob storage) already configured and running. In this blog, we change the output to use a Service Bus Queue instead. After that, we connect an Azure Function to this queue.

Create and connect a Service Bus Queue output

To create a Service Bus Queue, follow steps 1 and 2 of the .NET section in Get Started with Service Bus Queues. Now let's connect the queue to the Stream Analytics job that was created in the earlier fraud detection walk-through.

1. In the Azure portal, go to the **Outputs** blade of your job and select **Add** at the top of the page.

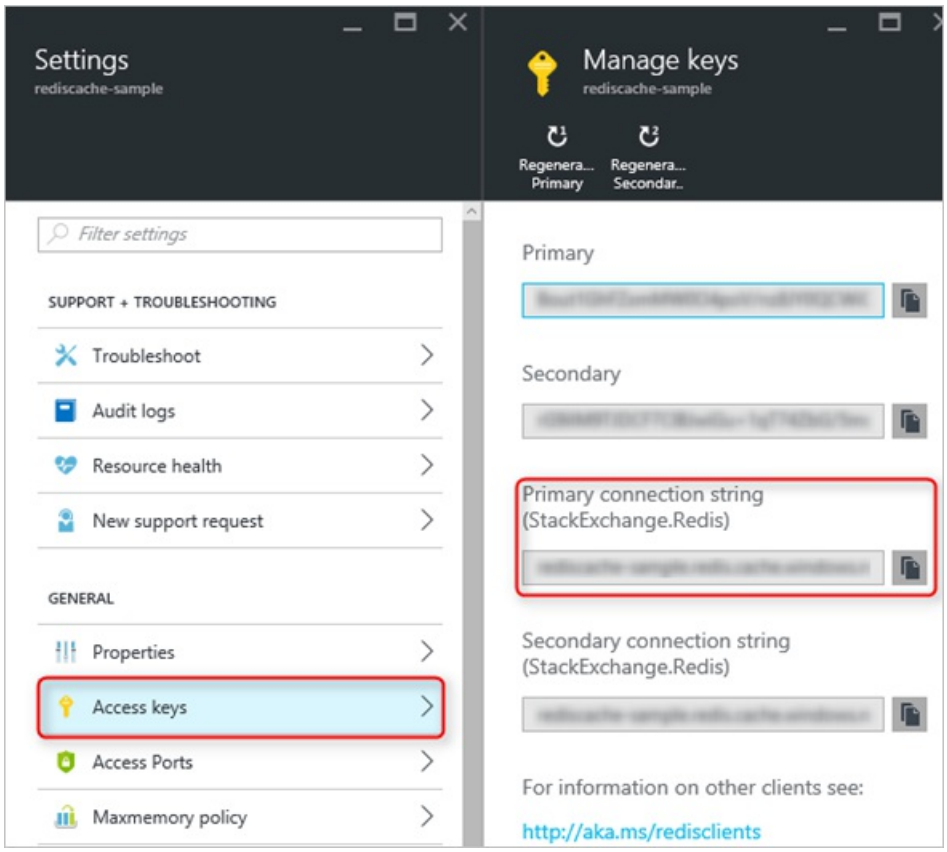


2. Choose **Service Bus Queue** as the **Sink** and follow the instructions on the screen. Be sure to choose the namespace of the Service Bus Queue you created in Get Started with Service Bus Queues. Click the "right" button when you are finished.
3. Specify the following values:
   - **Event Serializer Format**: JSON
   - **Encoding**: UTF8
   - **FORMAT**: Line separated
4. Click the **Create** button to add this source and to verify that Stream Analytics can successfully connect to the storage account.
5. In the **Query** tab, replace the current query with the following. Replace *[YOUR SERVICE BUS NAME] * with the output name you created in step 3.

```
SELECT
    System.Timestamp as Time, CS1.CallingIMSI, CS1.CallingNum as CallingNum1,
    CS2.CallingNum as CallingNum2, CS1.SwitchNum as Switch1, CS2.SwitchNum as Switch2

INTO [YOUR SERVICE BUS NAME]

FROM CallStream CS1 TIMESTAMP BY CallRecTime
JOIN CallStream CS2 TIMESTAMP BY CallRecTime
    ON CS1.CallingIMSI = CS2.CallingIMSI AND DATEDIFF(ss, CS1, CS2) BETWEEN 1 AND 5

WHERE CS1.SwitchNum != CS2.SwitchNum
```

Create an Azure Redis Cache

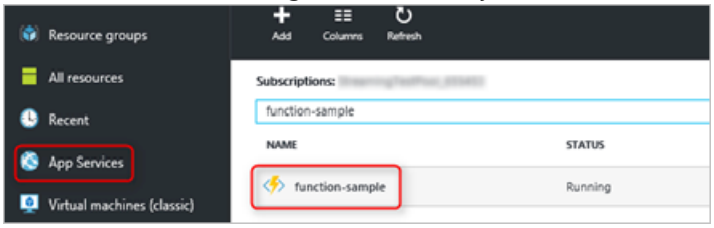
Create an Azure Redis cache by following the .NET section in How to Use Azure Redis Cache until the section called ***Configure the cache clients***. Once complete, you have a new Redis Cache. Under **All settings**, select **Access keys** and note down the ***Primary connection string***.



Create an Azure Function

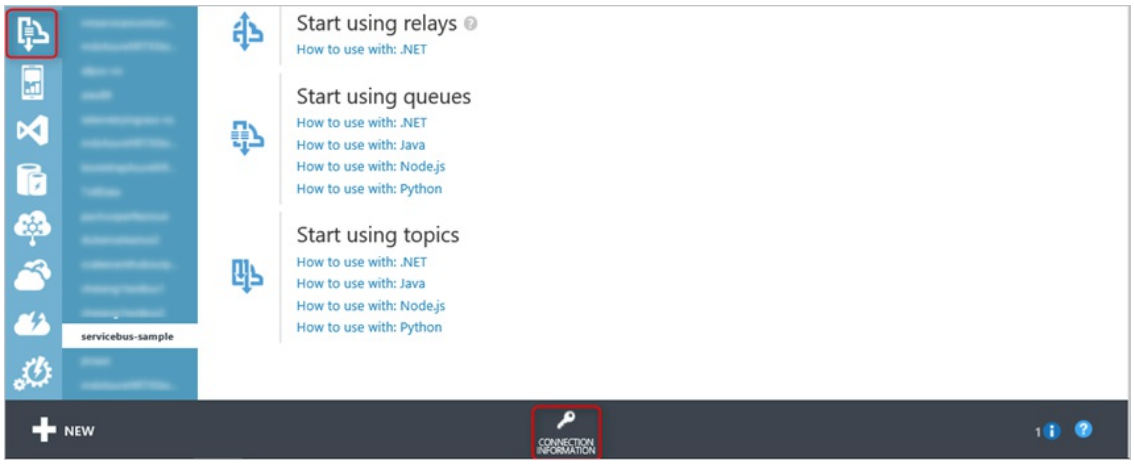
Follow Create your first Azure Function tutorial to get started with Azure Functions. If you already have an Azure function you would like to use, then skip ahead to Writing to Redis Cache

1. In the portal, select App Services from the left-hand navigation, then click your Azure function app name to get to the Function's app website.



2. Click **New Function > ServiceBusQueueTrigger – C#**. For the following fields, follow these instructions:

   - **Queue name**: The same name as the name you entered when you created the queue in Get Started with Service Bus Queues (not the name of the service bus). Make sure you use the queue that is connected to the Stream Analytics output.
   - **Service Bus connection**: Select **Add a connection string**. To find the connection string, go to the classic portal, select **Service Bus**, the service bus you created, and **CONNECTION INFORMATION** at the bottom of the screen. Make sure you are on the main screen on this page. Copy and paste the connection string. Feel free to enter any connection name.



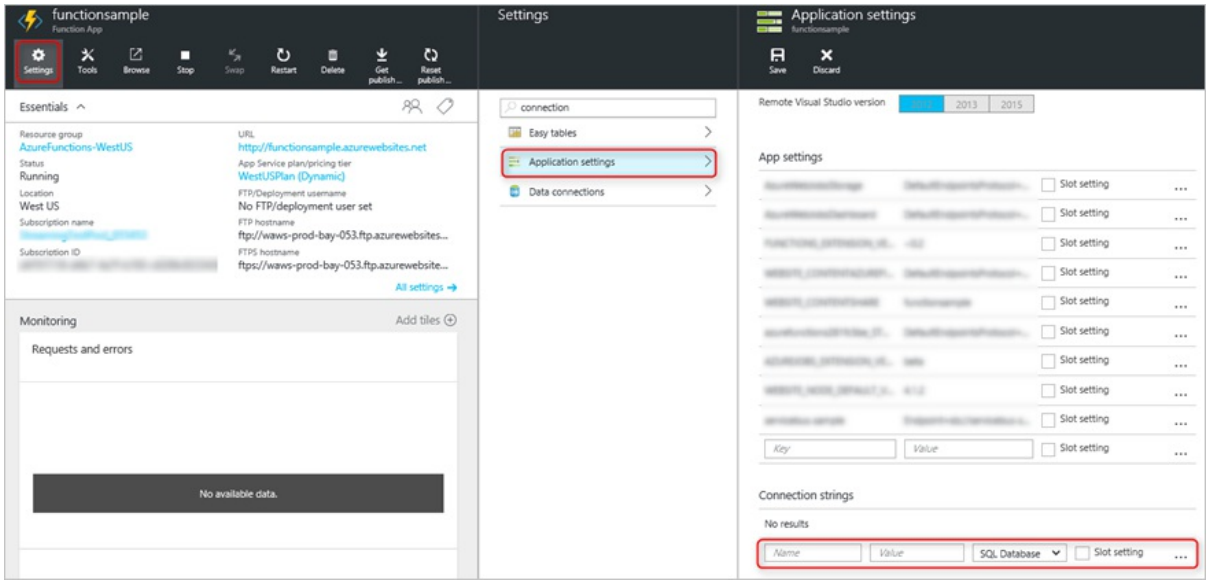
- **AccessRights**: Choose **Manage**

3. Click **Create**

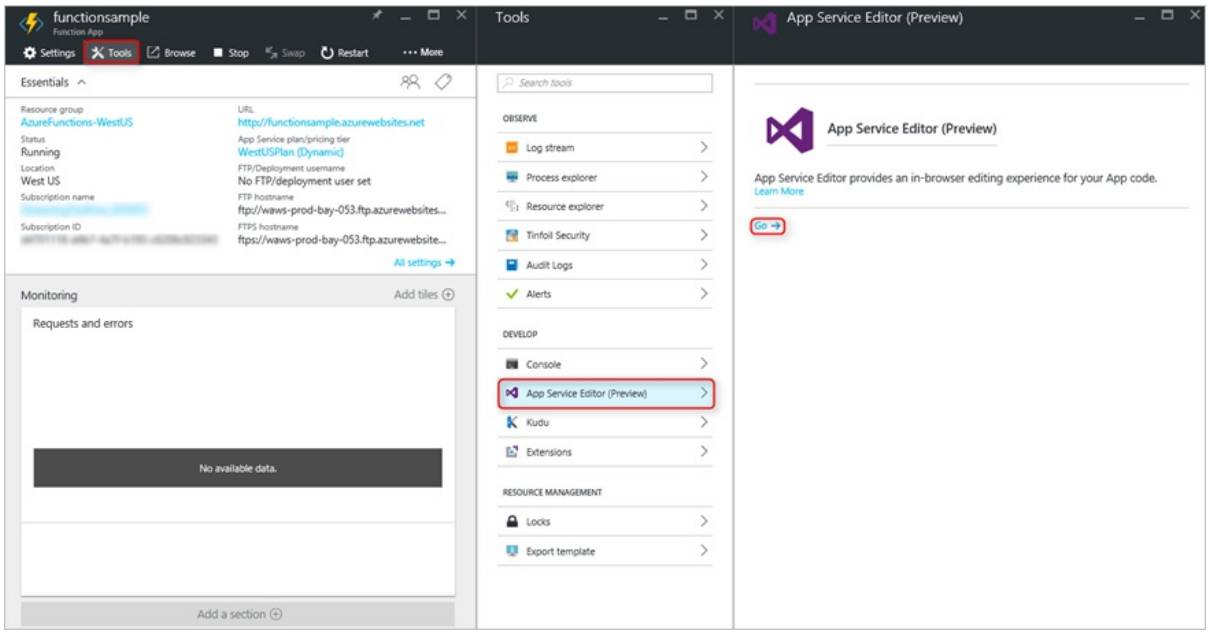
Writing to Redis Cache

We have now created an Azure Function that reads from a Service Bus Queue. All that is left to do is use our Function to write this data to the Redis Cache.

1. Select your newly created **ServiceBusQueueTrigger**, and click **Function app settings** on the top right corner. Select **Go to App Service Settings > Settings > Application settings**
2. In the Connection strings section, create a name in the **Name** section. Paste the primary connection string you found in the **Create a Redis Cache** step into the **Value** section. Select **Custom** where it says **SQL Database**.
3. Click **Save** at the top.



4. Now go back to the App Service Settings and select **Tools > App Service Editor (Preview) > On > Go**.



5. In an editor of your choice, create a JSON file named **project.json** with the following and save it to your local disk.

```
{
    "frameworks": {
        "net46": {
            "dependencies": {
                "StackExchange.Redis":"1.1.603",
                "Newtonsoft.Json": "9.0.1"
            }
        }
    }
}
```

6. Upload this file into the root directory of your function (not WWWROOT). You should see a file named **project.lock.json** automatically appear, confirming that the Nuget packages "StackExchange.Redis" and "Newtonsoft.Json" have been imported.
7. In the **run.csx** file, replace the pre-generated code with the following code. In the lazyConnection function, replace "CONN NAME" with the name you created in step 2 of **Store data into the Redis cache**.

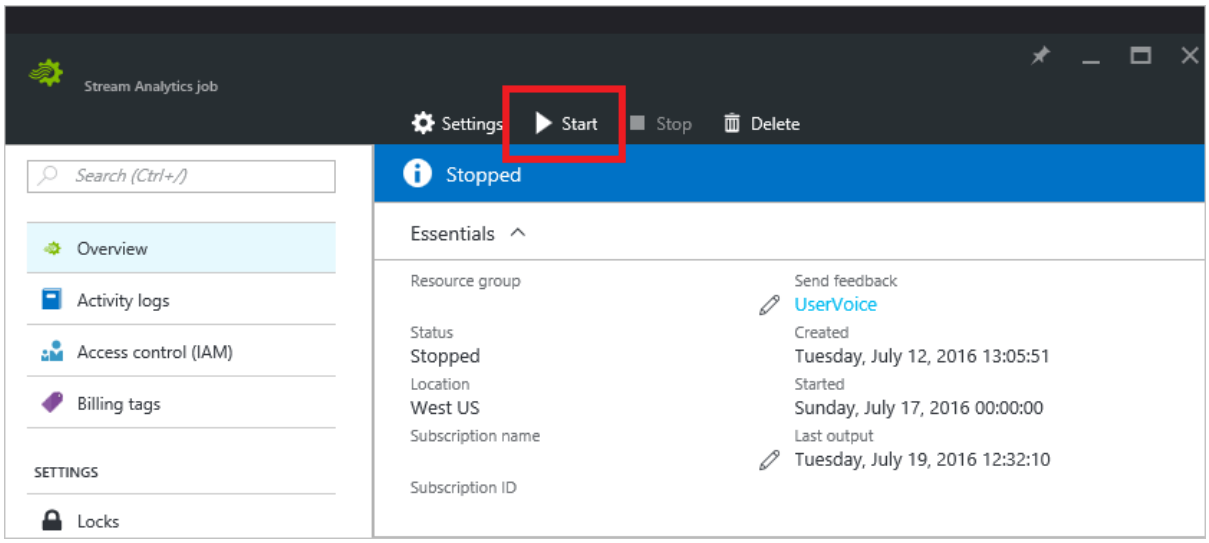
```
using System;
using System.Threading.Tasks;
using StackExchange.Redis;
using Newtonsoft.Json;
using System.Configuration;

public static void Run(string myQueueItem, TraceWriter log)
{
    log.Info($"Function processed message: {myQueueItem}");

    // Connection refers to a property that returns a ConnectionMultiplexer
    IDatabase db = Connection.GetDatabase();
    log.Info($"Created database {db}");

    // Parse JSON and extract the time
    var message = JsonConvert.DeserializeObject<dynamic>(myQueueItem);
    string time = message.time;
    string callingnum1 = message.callingnum1;

    // Perform cache operations using the cache object...
    // Simple put of integral data types into the cache
    string key = time + " - " + callingnum1;
    db.StringSet(key, myQueueItem);
    log.Info($"Object put in database. Key is {key} and value is {myQueueItem}");

    // Simple get of data types from the cache
    string value = db.StringGet(key);
    log.Info($"Database got: {value}");
}

// Connect to the Service Bus
private static Lazy<ConnectionMultiplexer> lazyConnection =
    new Lazy<ConnectionMultiplexer>(() =>
        {
            var cnn = ConfigurationManager.ConnectionStrings["CONN NAME"].ConnectionString
            return ConnectionMultiplexer.Connect();
        });

public static ConnectionMultiplexer Connection
{
    get
    {
        return lazyConnection.Value;
    }
}
```

Start the Stream Analytics job

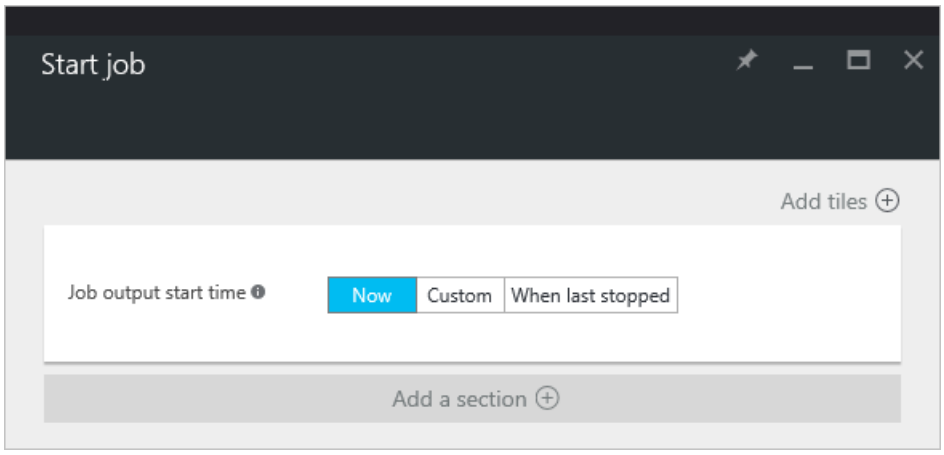
1. Start the telcodatagen.exe application. The usage is as follows:

   ```
   telcodatagen.exe [#NumCDRsPerHour] [SIM Card Fraud Probability] [#DurationHours]
   ```

2. From the Stream Analytics Job blade in the portal, click **Start** at the top of the page.



3. In the **Start job** blade that appears, select **Now** and then click the **Start** button at the bottom of the screen. The job status changes to Starting and after some time changes to Running.

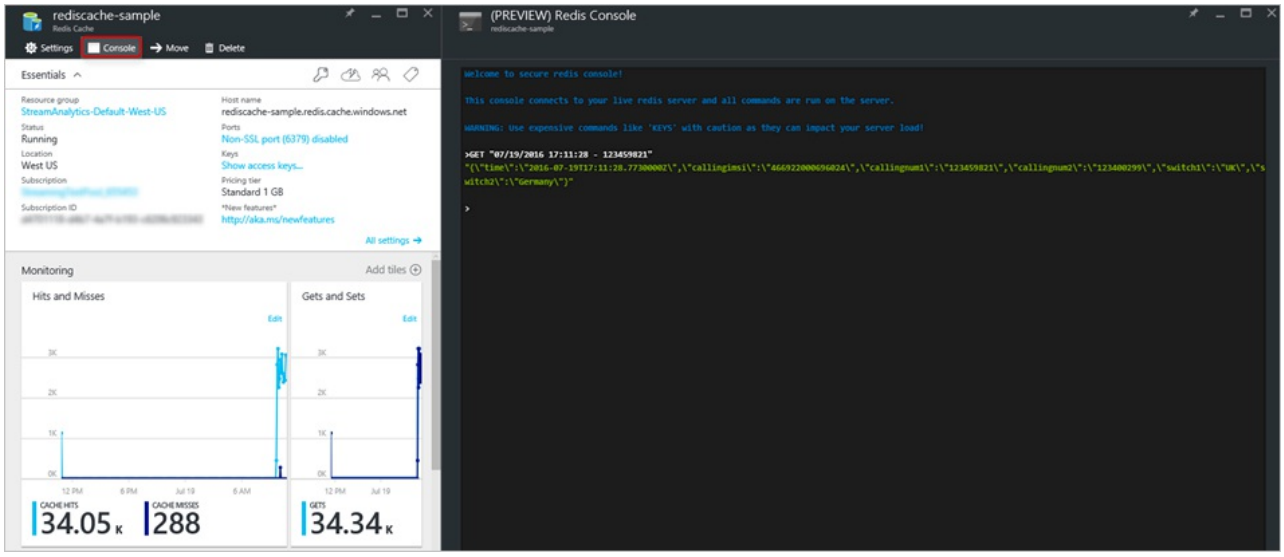


Run solution and check results

Going back to your **ServiceBusQueueTrigger** page, you should now see log statements. These logs show that you got something from the Service Bus Queue, put it into the database, and fetched it out using the time as the key!

To verify that your data is in your Redis cache, go to your Redis cache page in the new portal (as shown in the preceding Create an Azure Redis Cache step) and select Console.

Now you can write Redis commands to confirm that data is in fact in the cache.



Next steps

We're excited about the new things Azure Functions and Stream analytics can do together, and we hope this unlocks new possibilities for you. If you have any feedback on what you want next, feel free to use the Azure UserVoice site.

If you are new Microsoft Azure, we invite you to try it out by signing up for a free Azure trial account. If you are new to Stream Analytics, then we invite you to create your first Stream Analytics job.

If you need any help or have questions, post on MSDN or Stackoverflow forums.

You can also see the following resources:

- Azure Functions developer reference
- Azure Functions C# developer reference
- Azure Functions F# developer reference
- Azure Functions NodeJS developer reference
- Azure Functions triggers and bindings
- How to monitor Azure Redis Cache

To stay up-to-date on all the latest news and features, follow @AzureStreaming on Twitter.

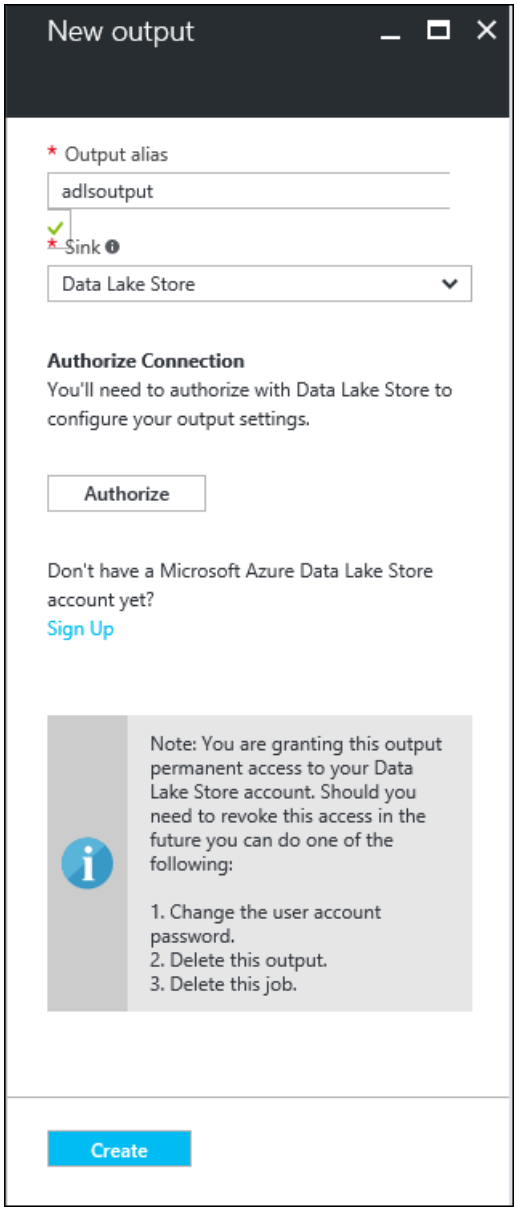
Stream Analytics Data Lake Store output

1/25/2017 • 3 min to read • Edit on GitHub

Stream Analytics jobs support several output methods, one being an Azure Data Lake Store. Azure Data Lake Store is an enterprise-wide hyper-scale repository for big data analytic workloads. Data Lake Store enables you to store data of any size, type and ingestion speed for operational and exploratory analytics.

Authorize a Data Lake Store account

1. When Data Lake Store is selected as an output in the Azure Management portal, you will be prompted to authorize the usage of your existing Data Lake Store or to request access to the Data Lake Store via the Azure Classic Portal.



2. If you already have access to Data Lake Store, click "Authorize Now" and for a brief time a page will pop up indicating "Redirecting to authorization..". The page will automatically close and you will be presented with the page that would allow you to configure the Data Lake Store output.

If you have not signed up for Data Lake Store, you can follow the "Sign up now" link to initiate the request, or follow the get started instructions.

Configure the Data Lake Store output properties

Once you have the Data Lake Store account authenticated, you can configure the properties for your Data Lake Store output. The table below is the list of property names and their description to configure your Data Lake Store output.

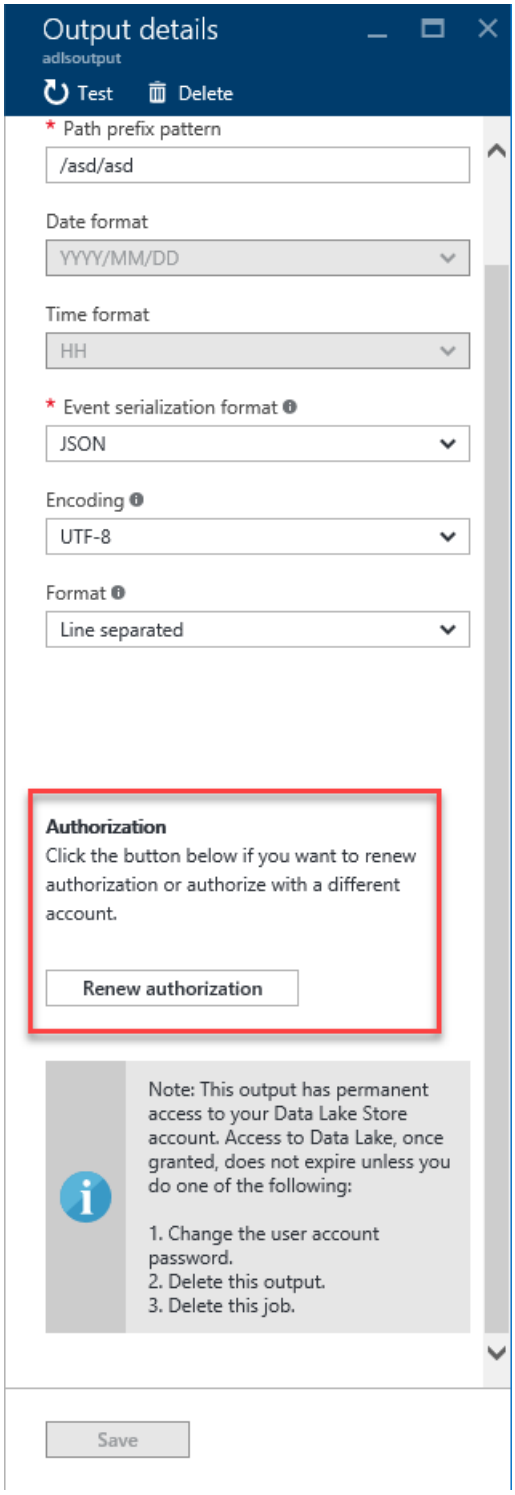
PROPERTY NAME	DESCRIPTION
Output Alias	This is a friendly name used in queries to direct the query output to this Data Lake Store.
Data Lake Store Account	The name of the storage account where you are sending your output. You will be presented with a drop down list of Data Lake Store accounts to which the user logged in to the portal has access to.
Path Prefix Pattern [*optional*]	The file path used to write your files within the specified Data Lake Store Account. {date}, {time} Example 1: folder1/logs/{date}/{time} Example 2: folder1/logs/{date}
Date Format [*optional*]	If the date token is used in the prefix path, you can select the date format in which your files are organized. Example: YYYY/MM/DD
Time Format [*optional*]	If the time token is used in the prefix path, specify the time format in which your files are organized. Currently the only supported value is HH.
Event Serialization Format	Serialization format for output data. JSON, CSV, and Avro are supported.
Encoding	If CSV or JSON format, an encoding must be specified. UTF-8 is the only supported encoding format at this time.
Delimiter	Only applicable for CSV serialization. Stream Analytics supports a number of common delimiters for serializing CSV data. Supported values are comma, semicolon, space, tab and vertical bar.
Format	Only applicable for JSON serialization. Line separated specifies that the output will be formatted by having each JSON object separated by a new line. Array specifies that the output will be formatted as an array of JSON objects.

Renew Data Lake Store Authorization

Currently, there is a limitation where the authentication token needs to be manually refreshed every 90 days for all jobs with Data Lake Store output. You will also need to re-authenticate your Data Lake Store account if you have changed your password since your job was created or last authenticated. A symptom of this issue is no job output and an error in the Operation Logs indicating need for re-authorization.

To resolve this issue, stop your running job and go to your Data Lake Store output. Click the "Renew authorization" link, and for a brief time a page will pop up indicating "Redirecting to authorization..". The page will automatically close and if successful, will indicate "Authorization has been successfully renewed". You then need to click "Save" at

the bottom of the page, and can proceed by restarting your job from the Last Stopped Time to avoid data loss.



Target Azure DocumentDB for JSON output from Stream Analytics

1/25/2017 • 4 min to read • Edit on GitHub

Stream Analytics can target Azure DocumentDB for JSON output, enabling data archiving and low-latency queries on unstructured JSON data. This document covers some best practices for implementing this configuration.

For those who are unfamiliar with DocumentDB, take a look at DocumentDB's learning path to get started.

Basics of DocumentDB as an output target

The Azure DocumentDB output in Stream Analytics enables writing your stream processing results as JSON output into your DocumentDB collection(s). Stream Analytics does not create collections in your database, instead requiring you to create them upfront. This is so that the billing costs of DocumentDB collections are transparent to you, and so that you can tune the performance, consistency and capacity of your collections directly using the DocumentDB APIs. We recommend using one DocumentDB Database per streaming job to logically separate your collections for a streaming job.

Some of the DocumentDB collection options are detailed below.

Tune consistency, availability, and latency

To match your application requirements, DocumentDB allows you to fine tune the Database and Collections and make trade-offs between consistency, availability and latency. Depending on what levels of read consistency your scenario needs against read and write latency, you can choose a consistency level on your database account. Also by default, DocumentDB enables synchronous indexing on each CRUD operation to your collection. This is another useful option to control the write/read performance in DocumentDB. For further information on this topic, review the change your database and query consistency levels article.

Choose a performance level

DocumentDB collections can be created at 3 different performance levels (S1, S2 or S3), which determine the throughput available for CRUDs to that collection. Additionally, performance is impacted by the indexing/consistency levels on your collection. Please refer to this article for understanding these performance levels in detail.

Upserts from Stream Analytics

Stream Analytics integration with DocumentDB allows you to insert or update records in your DocumentDB collection based on a given Document ID column. This is also referred to as an *Upsert*.

Stream Analytics utilizes an optimistic Upsert approach, where updates are only done when insert fails due to a Document ID conflict. This update is performed by Stream Analytics as a PATCH, so it enables partial updates to the document, i.e. addition of new properties or replacing an existing property is performed incrementally. Note that changes in the values of array properties in your JSON document result in the entire array getting overwritten, i.e. the array is not merged.

Data partitioning in DocumentDB

DocumentDB Partitioned Collections are supported now and is the recommended approach for partitioning your

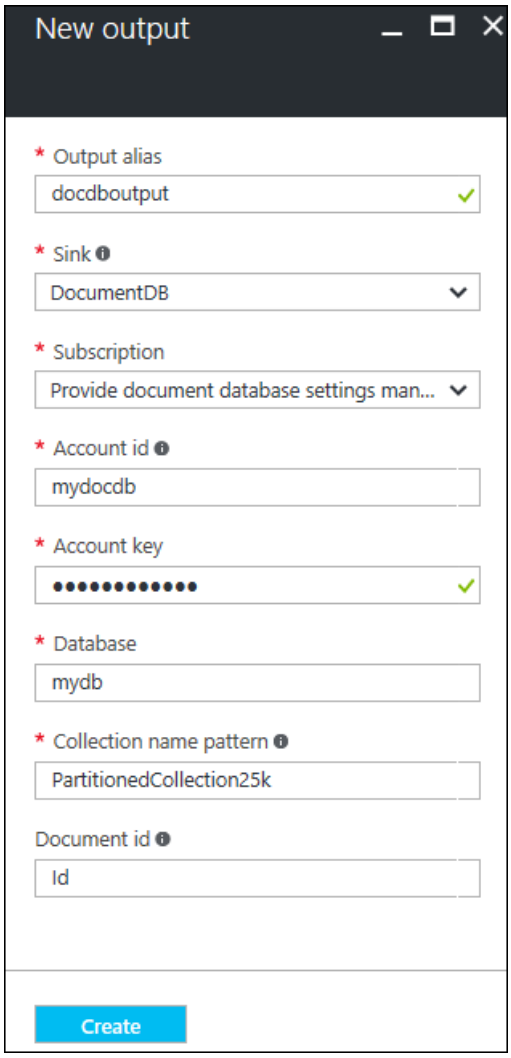
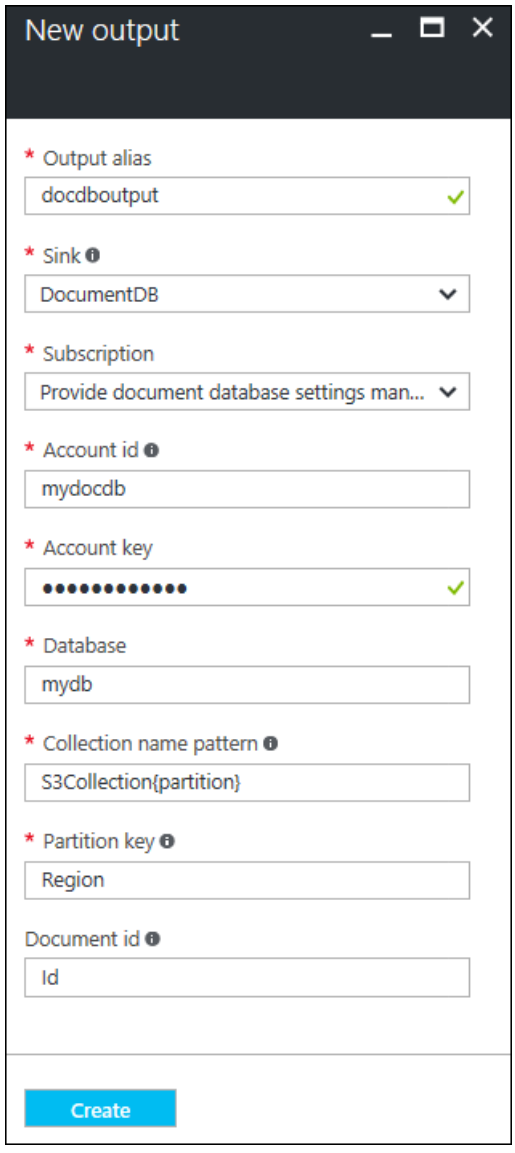
data.

For Single DocumentDB collections, Stream Analytics still allows you to partition your data based on both the query patterns and performance needs of your application. Each collection may contain up to 10GB of data (maximum) and currently there is no way to scale up (or overflow) a collection. For scaling out, Stream Analytics allows you to write to multiple collections with a given prefix (see usage details below). Stream Analytics uses the consistent Hash Partition Resolver strategy based on the user provided PartitionKey column to partition its output records. The number of collections with the given prefix at the streaming job's start time is used as the output partition count, to which the job writes to in parallel (DocumentDB Collections = Output Partitions). For a single S3 collection with lazy indexing doing only inserts, about 0.4 MB/s write throughput can be expected. Using multiple collections can allow you to achieve higher throughput and increased capacity.

If you intend to increase the partition count in the future, you may need to stop your job, repartition the data from your existing collections into new collections and then restart the Stream Analytics job. More details on using PartitionResolver and re-partitioning along with sample code, will be included in a follow-up post. The article Partitioning and scaling in DocumentDB also provides details on this.

DocumentDB settings for JSON output

Creating DocumentDB as an output in Stream Analytics generates a prompt for information as seen below. This section provides an explanation of the properties definition.

PARTITIONED COLLECTION	MULTIPLE "SINGLE PARTITION" COLLECTIONS
	

NOTE

The **Multiple "Single Partition" collections** scenario requires a partition key and is a supported configuration.

- **Output Alias** – An alias to refer this output in your ASA query
- **Account Name** – The name or endpoint URI of the DocumentDB account.
- **Account Key** – The shared access key for the DocumentDB account.
- **Database** – The DocumentDB database name.
- **Collection Name Pattern** – The collection name or their pattern for the collections to be used. The collection name format can be constructed using the optional {partition} token, where partitions start from 0. Following are sample valid inputs:
  1) MyCollection – One collection named "MyCollection" must exist.
  2) MyCollection{partition} – Such collections must exist– "MyCollection0", "MyCollection1", "MyCollection2" and so on.
- **Partition Key** – Optional. This is only needed if you are using a {parition} token in your collection name pattern. The name of the field in output events used to specify the key for partitioning output across collections. For single collection output, any arbitrary output column can be used e.g. PartitionId.
- **Document ID** – Optional. The name of the field in output events used to specify the primary key on which insert or update operations are based.

Stream Analytics & Power BI: A real-time analytics dashboard for streaming data

2/1/2017 • 6 min to read • Edit on GitHub

Azure Stream Analytics allows you to take advantage of one of the leading business intelligence tools, Microsoft Power BI. Learn how to use Azure Stream Analytics to analyze high-volume, streaming data and get the insight in a real-time Power BI analytics dashboard.

Use Microsoft Power BI to build a live dashboard quickly. Watch a video illustrating the scenario.

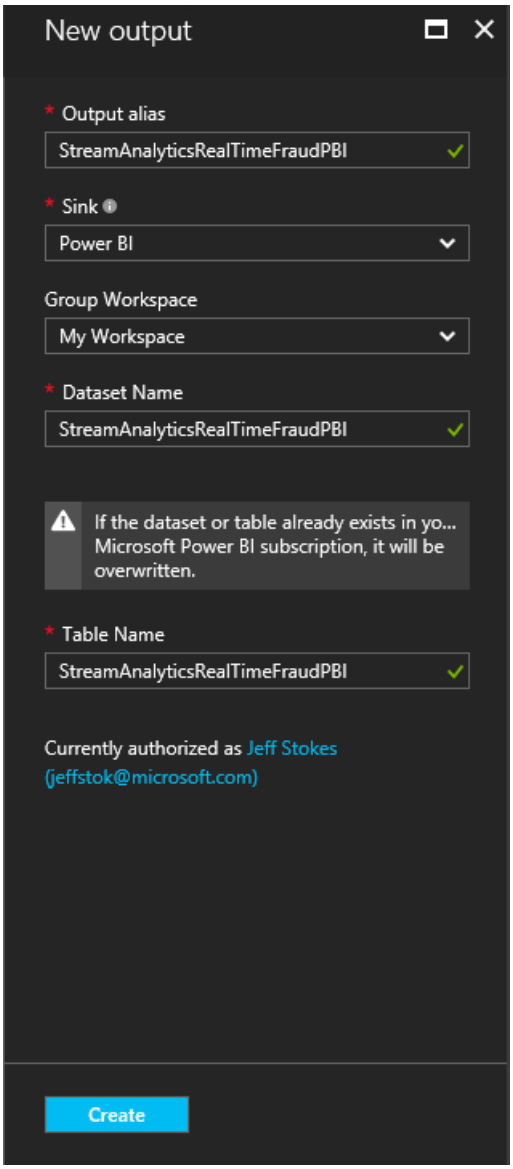
In this article, learn how create your own custom business intelligence tools by using Power BI as an output for your Azure Stream Analytics jobs and utilize a real-time dashboard.

Prerequisites

- Microsoft Azure Account
- Work or school account for Power BI
- Complete the Get Started scenario Real-time fraud detection. This article builds upon that workflow and adds a Power BI streaming dataset output.

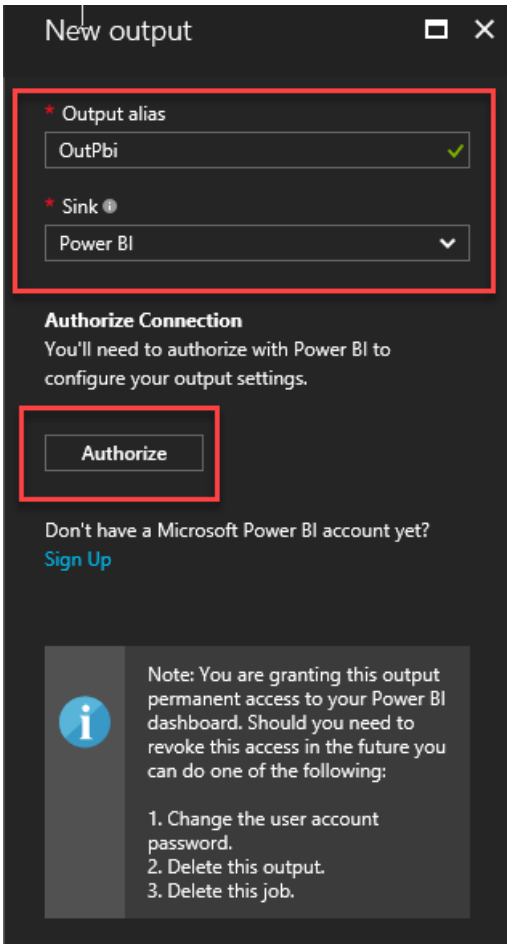
Add Power BI output

Now that an input exists for the job, an output to Power BI can be defined. Select the box in the middle of the job dashboard **Outputs**. Then click the familiar **+ Add** button and create your output.

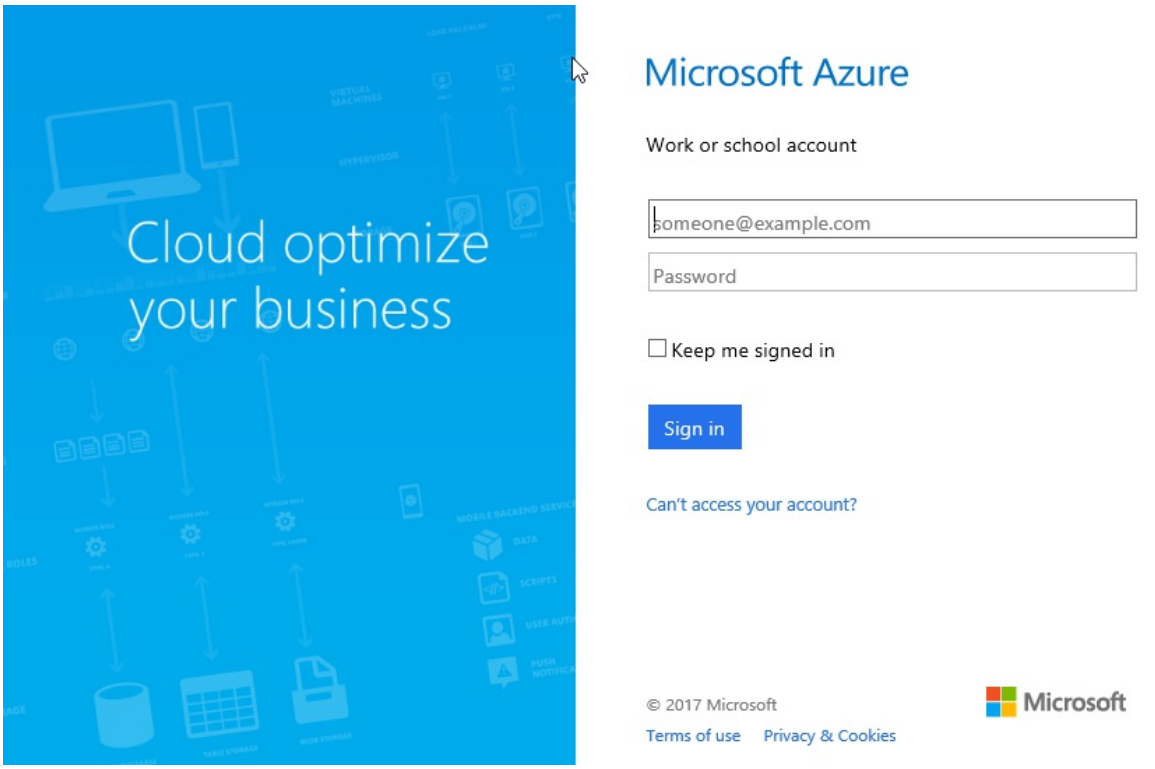


Provide the **Output Alias** – You can put any output alias that is easy for you to refer to. This output alias is particularly helpful if you decide to have multiple outputs for your job. In that case, you have to refer to this output in your query. For example, let's use the output alias value = "StreamAnalyticsRealTimeFraudPBI".

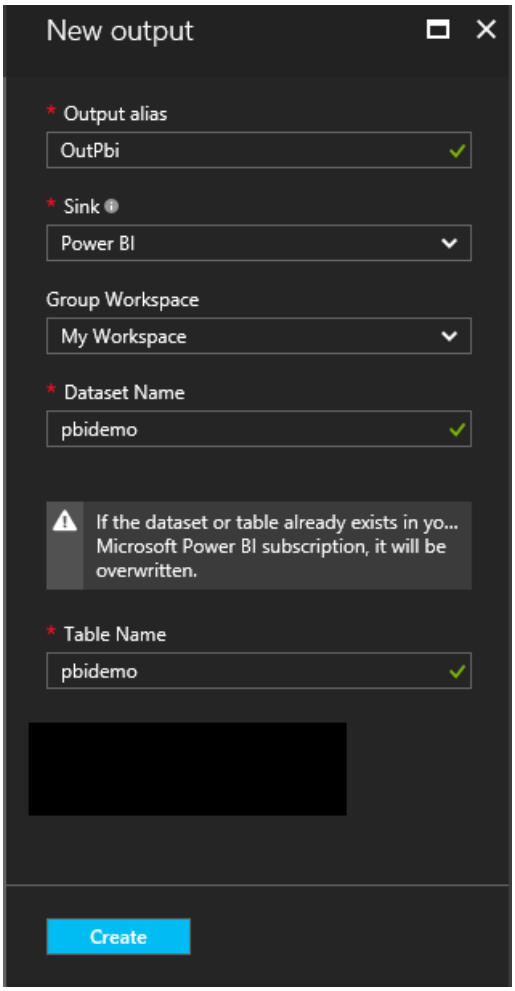
Then click the **Authorize** button.



This will prompt a window to provide your Azure credentials (work or school account) and it will provide your Azure job access to your Power BI area.



The authorization window will disappear when satisfied and the New output area will have fields for the Dataset Name and Table Name.



Define them as follows:

- **Group Workspace** – Select a workspace in your Power BI tenant under which the dataset will be created.
- **Dataset Name** - Provide a dataset name that you want your Power BI output to have. For example, let's use "StreamAnalyticsRealTimeFraudPBI".
- **Table Name** - Provide a table name under the dataset of your Power BI output. Let's say we call it "StreamAnalyticsRealTimeFraudPBI". Currently, Power BI output from Stream Analytics jobs may only have one table in a dataset.

Click **Create** and now you output configuration is complete.

> **WARNING**
> Please be aware that if Power BI already had a dataset and table with the same name as the one you provided in this Stream Analytics job, the existing data will be overwritten! Also, you should not explicitly create this dataset and table in your Power BI account. They will be automatically created when you start your Stream Analytics job and the job starts pumping output into Power BI. If your job query doesn't return any results, the dataset and table will not be created.

The dataset is created with the following settings set;

- defaultRetentionPolicy: BasicFIFO - data is FIFO, 200k maximum rows
- defaultMode: pushStreaming: supports both streaming tiles and traditional report-based visuals (aka push)
- creating datasets with other flags is unsupported at this time

For more information on Power BI datasets see the Power BI REST API reference.

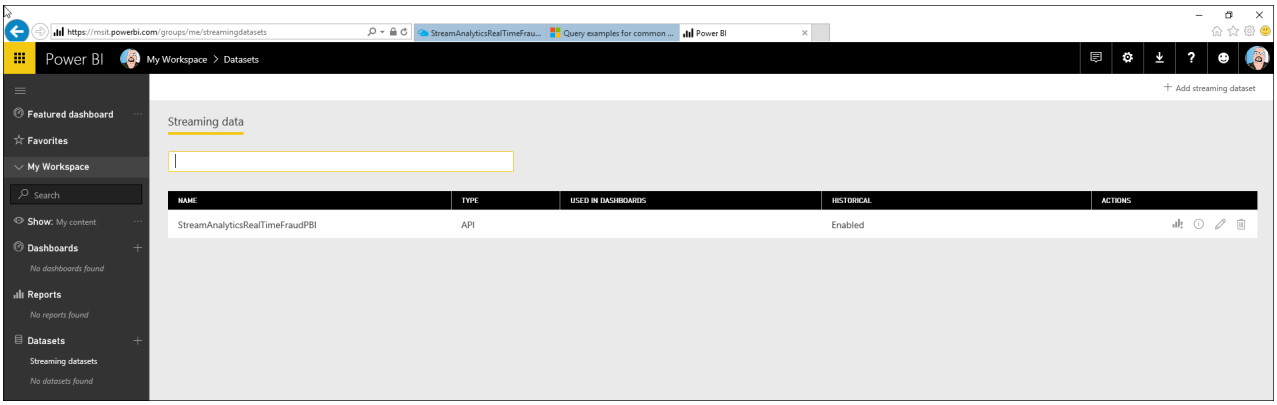
Write query

Go to the **Query** tab of your job. Write your query, the output of which you want in your Power BI. For example, it could be something such as the following SQL query to catch SIM fraud in the telecommunications industry:

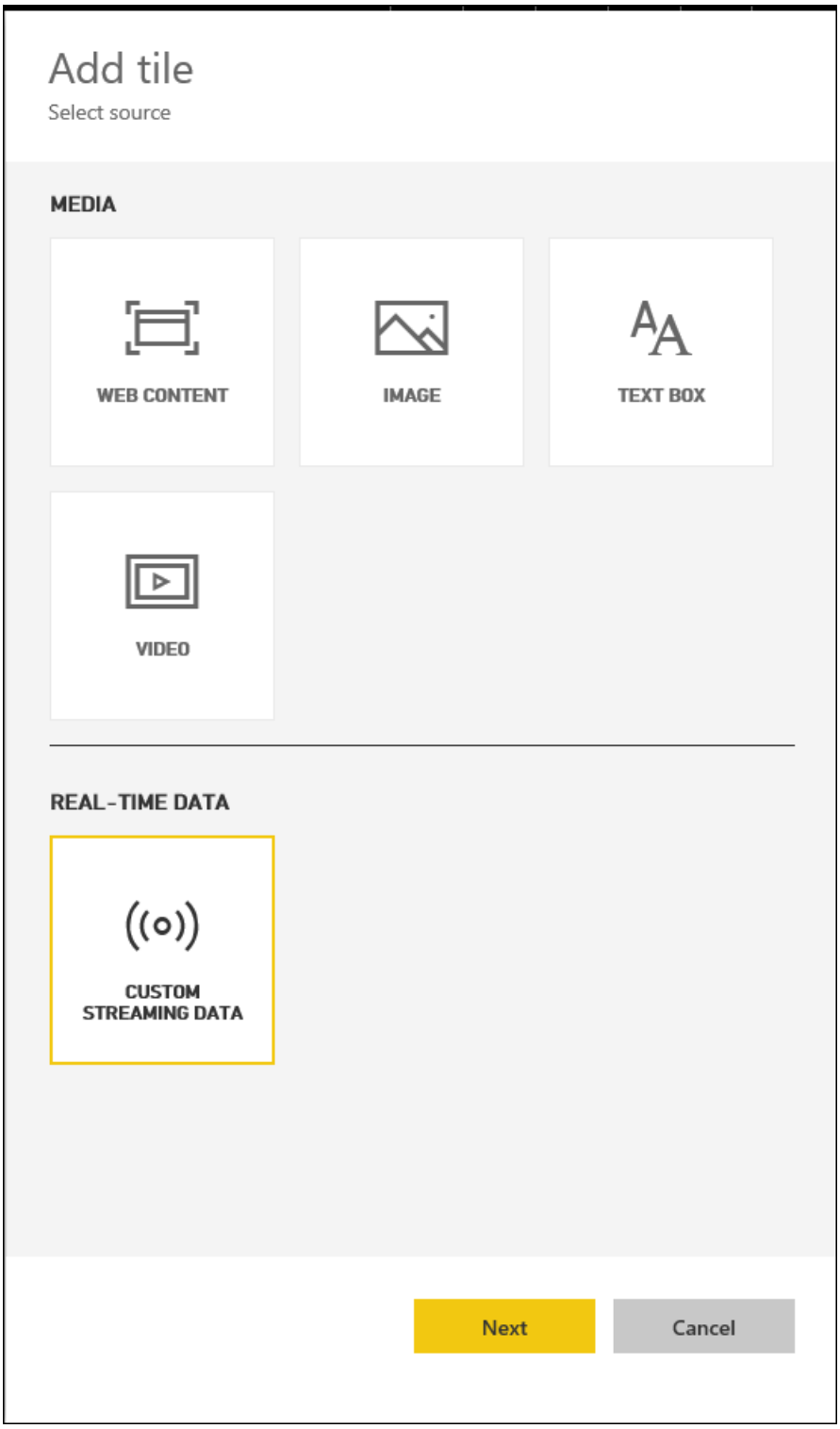
```
/* Our criteria for fraud:
 Calls made from the same caller to two phone switches in different locations (e.g. Australia and Europe)
within 5 seconds) */

 SELECT System.Timestamp AS WindowEnd, COUNT(*) AS FraudulentCalls
 INTO "StreamAnalyticsRealTimeFraudPBI"
 FROM "StreamAnalyticsRealTimeFraudInput" CS1 TIMESTAMP BY CallRecTime
 JOIN "StreamAnalyticsRealTimeFraudInput" CS2 TIMESTAMP BY CallRecTime

/* Where the caller is the same, as indicated by IMSI (International Mobile Subscriber Identity) */
 ON CS1.CallingIMSI = CS2.CallingIMSI

/* ...and date between CS1 and CS2 is between 1 and 5 seconds */
 AND DATEDIFF(ss, CS1, CS2) BETWEEN 1 AND 5

/* Where the switch location is different */
 WHERE CS1.SwitchNum != CS2.SwitchNum
 GROUP BY TumblingWindow(Duration(second, 1))
```

Create the dashboard in Power BI

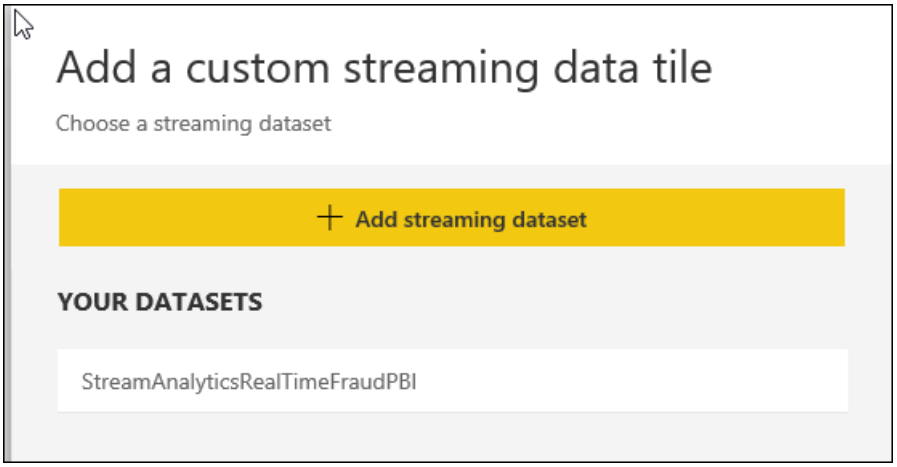
Go to Powerbi.com and login with your work or school account. If the Stream Analytics job query outputs results, you will see your dataset is already created:



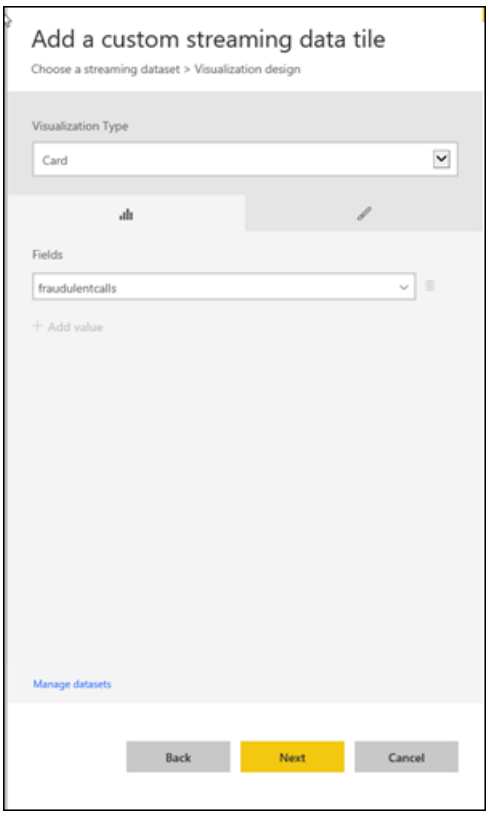
Click Add tile and select your custom streaming data.



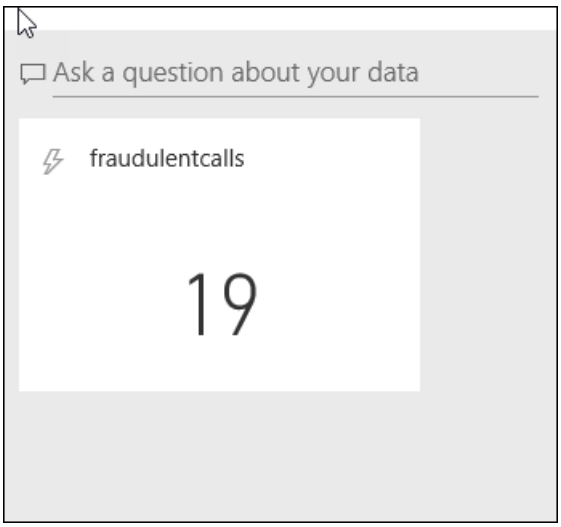
Then select your dataset from the list:



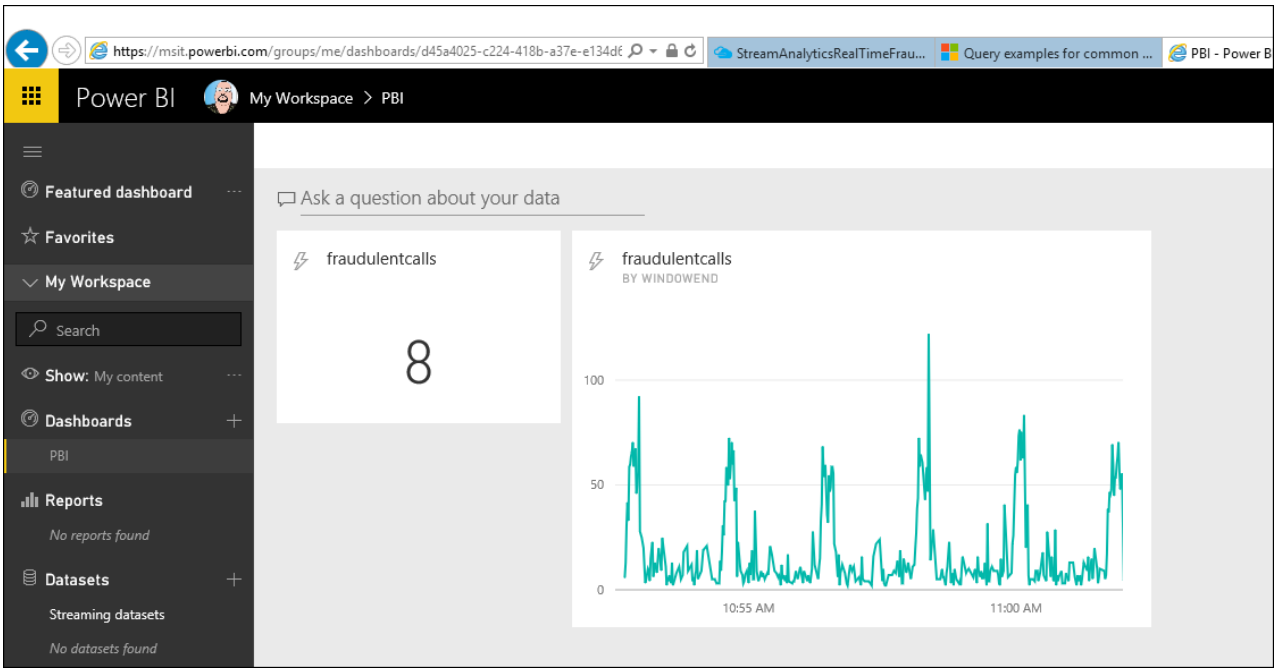
Now create a visualization card and select the field fraudulentcalls.



Now we have a fraud counter!



Walk through the exercise of adding a tile again but this time select line chart. Add fraudulentcalls as the value and the axis as windowend. I selected the last 10 minutes:



Note that this tutorial demonstrated how to create but one kind of chart for a dataset. Power BI can help you create other customer business intelligence tools for your organization. For another example of a Power BI dashboard, watch the Getting Started with Power BI video.

For further information on configuring a Power BI output and to utilize Power BI groups, review the Power BI section of Understanding Stream Analytics outputs. Another helpful resource to learn more about creating Dashboards with Power BI is Dashboards in Power BI.

Limitations and best practices

Power BI employs both concurrency and throughput constraints as described here: https://powerbi.microsoft.com/pricing

Currently, you Power BI can be called roughly once per second. Streaming visuals support packets of size 15kb. Beyond that and streaming visuals will fail (but push will continue to work).

Because of those Power BI lands itself most naturally to cases where Azure Stream Analytics does a significant data load reduction. We recommend using TumblingWindow or HoppingWindow to ensure that data push would be at most 1 push/second and that your query lands within the throughput requirements – you can use the following equation to compute the value to give your window in seconds:

$$\left\lceil \frac{EntityCount * 60 * 60}{Throughput} \right\rceil$$

As an example – If you have 1,000 devices sending data every second, you are on the Power BI Pro SKU that supports 1,000,000 rows/hour and you want to get average data per device on Power BI you can do at most a push every 4 seconds per device (as shown below):

$$\left\lceil \frac{EntityCount * 60 * 60}{Throughput} \right\rceil = \left\lceil \frac{1{,}000 * 60 * 60}{1{,}000{,}000} \right\rceil = \lceil 3.6 \rceil = 4$$

Which means we would change the original query to:

```
SELECT
    MAX(hmdt) AS hmdt,
    MAX(temp) AS temp,
    System.TimeStamp AS time,
    dspl
INTO
    OutPBI
FROM
    Input TIMESTAMP BY time
GROUP BY
    TUMBLINGWINDOW(ss,4),
    dspl
```

Renew authorization

You need to re-authenticate your Power BI account if its password has changed since your job was created or last authenticated. If Multi-Factor Authentication (MFA) is configured on your Azure Active Directory (AAD) tenant you will also need to renew Power BI authorization every 2 weeks. A symptom of this issue is no job output and an "Authenticate user error" in the Operation Logs.

Similarly, if a job attempts to start while the token is expired, an error will occur and the job start will fail. To resolve this issue, stop your running job and go to your Power BI output. Click the “Renew authorization” link, and restart your job from the Last Stopped Time to avoid data loss. Once the authorization is refreshed with Power BI you will see a green alert in the authorization area to reflect the issue is resolved.

Get help

For further assistance, try our Azure Stream Analytics forum

Next steps

- Introduction to Azure Stream Analytics
- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Azure Stream Analytics JavaScript user-defined functions

2/8/2017 • 3 min to read • Edit on GitHub

Azure Stream Analytics supports user-defined functions written in JavaScript. With the rich set of **String**, **RegExp**, **Math**, **Array**, and **Date** methods that JavaScript provides, complex data transformations with Stream Analytics jobs become easier to create.

JavaScript user-defined functions

JavaScript user-defined functions support stateless, compute-only scalar functions that do not require external connectivity. The return value of a function can only be a scalar (single) value. After you add a JavaScript user-defined function to a job, you can use the function anywhere in the query, like a built-in scalar function.

Here are some 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

Although functions like **Date.GetDate()** or **Math.random()** are not blocked in the functions definition, you should avoid using them. These functions **do not** return the same result every time you call them, and the Azure Stream Analytics service does not keep a journal of function invocations and returned results. If a function returns different result on the same events, repeatability is not guaranteed when a job is restarted by you or by the Stream Analytics service.

Add a JavaScript user-defined function in the Azure portal

To create a simple JavaScript user-defined function under an existing Stream Analytics job, do these steps:

1. In the Azure portal, find your Stream Analytics job.
2. Under **JOB TOPOLOGY**, select your function. An empty list of functions appears.
3. To create a new user-defined function, select **Add**.
4. On the **New Function** blade, for **Function Type**, select **JavaScript**. A default function template appears in the editor.
5. For the **UDF alias**, enter **hex2Int**, and change the function implementation as follows:

```
// Convert Hex value to integer.
function main(hexValue) {
  return parseInt(hexValue, 16);
}
```

6. Select **Save**. Your function appears in the list of functions.

7. Select the new **hex2Int** function, and check the function definition. All functions have a **UDF** prefix added to the function alias. You need to *include the prefix* when you call the function in your Stream Analytics query. In this case, you call **UDF.hex2Int**.

Call a JavaScript user-defined function in a query

1. In the query editor, under **JOB TOPOLOGY**, select **Query**.
2. Edit your query, and then call the user-defined function, like this:

```
SELECT
  time,
  UDF.hex2Int(offset) AS IntOffset
INTO
  output
FROM
  InputStream
```

3. To upload the sample data file, right-click the job input.

4. To test your query, select **Test**.

Supported JavaScript objects

Azure Stream Analytics JavaScript user-defined functions support standard, built-in JavaScript objects. For a list of these objects, see Global Objects.

Stream Analytics and JavaScript type conversion

There are differences in the types that the Stream Analytics query language and JavaScript support. This table lists the conversion mappings between the two:

STREAM ANALYTICS	JAVASCRIPT
bigint	Number (JavaScript can only represent integers up to precisely 2^53)
DateTime	Date (JavaScript only supports milliseconds)
double	Number
nvarchar(MAX)	String
Record	Object
Array	Array
NULL	Null

Here are JavaScript-to-Stream Analytics conversions:

JAVASCRIPT	STREAM ANALYTICS
Number	Bigint (if the number is round and between long.MinValue and long.MaxValue; otherwise, it's double)
Date	DateTime
String	nvarchar(MAX)
Object	Record
Array	Array
Null, Undefined	NULL
Any other type (for example, a function or error)	Not supported (results in runtime error)

Troubleshooting

JavaScript runtime errors are considered fatal, and are surfaced through the Activity log. To retrieve the log, in the Azure portal, go to your job and select **Activity log**.

Other JavaScript user-defined function patterns

Write nested JSON to output

If you have a follow-up processing step that uses a Stream Analytics job output as input, and it requires a JSON format, you can write a JSON string to output. The next example calls the **JSON.stringify()** function to pack all name/value pairs of the input, and then write them as a single string value in output.

JavaScript user-defined function definition:

```
function main(x) {
return JSON.stringify(x);
}
```

Sample query:

```
SELECT
    DataString,
    DataValue,
    HexValue,
    UDF.json_stringify(input) As InputEvent
INTO
    output
FROM
    input PARTITION BY PARTITIONID
```

Get help

For additional help, try our Azure Stream Analytics forum.

Next steps

- Introduction to Azure Stream Analytics

- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics query language reference
- Azure Stream Analytics management REST API reference

Machine Learning integration in Stream Analytics

1/23/2017 • 4 min to read • Edit on GitHub

Stream Analytics supports user-defined functions that call out to Azure Machine Learning endpoints. REST API support for this feature is detailed in the Stream Analytics REST API library. This article provides supplemental information needed for successful implementation of this capability in Stream Analytics. A tutorial has also been posted and is available here.

Overview: Azure Machine Learning terminology

Microsoft Azure Machine Learning provides a collaborative, drag-and-drop tool you can use to build, test, and deploy predictive analytics solutions on your data. This tool is called the *Azure Machine Learning Studio*. The studio is used to interact with the Machine Learning resources and easily build, test, and iterate on your design. These resources and their definitions are below.

- **Workspace**: The *workspace* is a container that holds all other Machine Learning resources together in a container for management and control.
- **Experiment**: *Experiments* are created by data scientists to utilize datasets and train a machine learning model.
- **Endpoint**: *Endpoints* are the Azure Machine Learning object used to take features as input, apply a specified machine learning model and return scored output.
- **Scoring Webservice**: A *scoring webservice* is a collection of endpoints as mentioned above.

Each endpoint has apis for batch execution and synchronous execution. Stream Analytics uses synchronous execution. The specific service is named a Request/Response Service in AzureML studio.

Machine Learning resources needed for Stream Analytics jobs

For the purposes of Stream Analytics job processing, a Request/Response endpoint, an apikey, and a swagger definition are all necessary for successful execution. Stream Analytics has an additional endpoint that constructs the url for swagger endpoint, looks up the interface and returns a default UDF definition to the user.

Configure a Stream Analytics and Machine Learning UDF via REST API

By using REST APIs you may configure your job to call Azure Machine Language functions. The steps are as follows:

1. Create a Stream Analytics job
2. Define an input
3. Define an output
4. Create a user-defined function (UDF)
5. Write a Stream Analytics transformation that calls the UDF
6. Start the job

Creating a UDF with basic properties

As an example, the following sample code creates a scalar UDF named *newudf* that binds to an Azure Machine Learning endpoint. Note that the *endpoint* (service URI) can be found on the API help page for the chosen service and the *apiKey* can be found on the Services main page.

```
    PUT :
/subscriptions/<subscriptionId>/resourceGroups/<resourceGroup>/providers/Microsoft.StreamAnalytics/streamingjob
s/<streamingjobName>/functions/<udfName>?api-version=<apiVersion>
```

Example request body:

```
    {
        "name": "newudf",
        "properties": {
            "type": "Scalar",
            "properties": {
                "binding": {
                    "type": "Microsoft.MachineLearning/WebService",
                    "properties": {
                        "endpoint":
"https://ussouthcentral.services.azureml.net/workspaces/f80d5d7a77fb4b46bf2a30c63c078dca/services/b7be5e40fd194
258796fb402c1958eaf/execute ",
                        "apiKey": "replacekeyhere"
                    }
                }
            }
        }
    }
```

Call RetrieveDefaultDefinition endpoint for default UDF

Once the skeleton UDF is created the complete definition of the UDF is needed. The RetreiveDefaultDefinition endpoint helps you get the default definition for a scalar function that is bound to an Azure Machine Learning endpoint. The payload below requires you to get the default UDF definition for a scalar function that is bound to an Azure Machine Learning endpoint. It doesn't specify the actual endpoint as it has already been provided during PUT request. Stream Analytics calls the endpoint provided in the request if it is provided explicitly. Otherwise it uses the one originally referenced. Here the UDF takes a single string parameter (a sentence) and returns a single output of type string which indicates the "sentiment" label for that sentence.

```
POST :
/subscriptions/<subscriptionId>/resourceGroups/<resourceGroup>/providers/Microsoft.StreamAnalytics/streamingjob
s/<streamingjobName>/functions/<udfName>/RetrieveDefaultDefinition?api-version=<apiVersion>
```

Example request body:

```
    {
        "bindingType": "Microsoft.MachineLearning/WebService",
        "bindingRetrievalProperties": {
            "executeEndpoint": null,
            "udfType": "Scalar"
        }
    }
```

A sample output of this would look something like below.

```
    {
        "name": "newudf",
        "properties": {
            "type": "Scalar",
            "properties": {
                "inputs": [{
                    "dataType": "nvarchar(max)",
                    "isConfigurationParameter": null
                }],
                "output": {
                    "dataType": "nvarchar(max)"
                },
                "binding": {
                    "type": "Microsoft.MachineLearning/WebService",
                    "properties": {
                        "endpoint":
"https://ussouthcentral.services.azureml.net/workspaces/f80d5d7a77ga4a4bbf2a30c63c078dca/services/b7be5e40fd194
258896fb602c1858eaf/execute",
                        "apiKey": null,
                        "inputs": {
                            "name": "input1",
                            "columnNames": [{
                                "name": "tweet",
                                "dataType": "string",
                                "mapTo": 0
                            }]
                        },
                        "outputs": [{
                            "name": "Sentiment",
                            "dataType": "string"
                        }],
                        "batchSize": 10
                    }
                }
            }
        }
    }
```

Patch UDF with the response

Now the UDF must be patched with the previous response, as shown below.

```
PATCH :
/subscriptions/<subscriptionId>/resourceGroups/<resourceGroup>/providers/Microsoft.StreamAnalytics/streamingjob
s/<streamingjobName>/functions/<udfName>?api-version=<apiVersion>
```

Request Body (Output from RetrieveDefaultDefinition):

```
    {
        "name": "newudf",
        "properties": {
            "type": "Scalar",
            "properties": {
                "inputs": [{
                    "dataType": "nvarchar(max)",
                    "isConfigurationParameter": null
                }],
                "output": {
                    "dataType": "nvarchar(max)"
                },
                "binding": {
                    "type": "Microsoft.MachineLearning/WebService",
                    "properties": {
                        "endpoint":
"https://ussouthcentral.services.azureml.net/workspaces/f80d5d7a77ga4a4bbf2a30c63c078dca/services/b7be5e40fd194
258896fb602c1858eaf/execute",
                        "apiKey": null,
                        "inputs": {
                            "name": "input1",
                            "columnNames": [{
                                "name": "tweet",
                                "dataType": "string",
                                "mapTo": 0
                            }]
                        },
                        "outputs": [{
                            "name": "Sentiment",
                            "dataType": "string"
                        }],
                        "batchSize": 10
                    }
                }
            }
        }
    }
```

Implement Stream Analytics transformation to call the UDF

Now query the UDF (here named scoreTweet) for every input event and write a response for that event to an output.

```
    {
        "name": "transformation",
        "properties": {
            "streamingUnits": null,
            "query": "select *,scoreTweet(Tweet) TweetSentiment into blobOutput from blobInput"
        }
    }
```

Get help

For further assistance, try our Azure Stream Analytics forum

Next steps

- Introduction to Azure Stream Analytics
- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs

- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Management .NET SDK: Set up and run analytics jobs using the Azure Stream Analytics API for .NET

1/25/2017 • 7 min to read • Edit on GitHub

Learn how to set up an run analytics jobs using the Stream Analytics API for .NET using the Management .NET SDK. Set up a project, create input and output sources, transformations, and start and stop jobs. For your analytics jobs, you can stream data from Blob storage or from an event hub.

See the management reference documentation for the Stream Analytics API for .NET.

Azure Stream Analytics is a fully managed service providing low-latency, highly available, scalable, complex event processing over streaming data in the cloud. Stream Analytics enables customers to set up streaming jobs to analyze data streams, and allows them to drive near real-time analytics.

Prerequisites

Before you begin this article, you must have the following:

- Install Visual Studio 2012 or 2013.
- Download and install Azure .NET SDK.
- Create an Azure Resource Group in your subscription. The following is a sample Azure PowerShell script. For Azure PowerShell information, see Install and configure Azure PowerShell;

```
  # Log in to your Azure account
  Add-AzureAccount

  # Select the Azure subscription you want to use to create the resource group
  Select-AzureSubscription -SubscriptionName <subscription name>

      # If Stream Analytics has not been registered to the subscription, remove the remark symbol (#) to
run the Register-AzureRMProvider cmdlet to register the provider namespace
      #Register-AzureRMProvider -Force -ProviderNamespace 'Microsoft.StreamAnalytics'

  # Create an Azure resource group
  New-AzureResourceGroup -Name <YOUR RESOURCE GROUP NAME> -Location <LOCATION>
```

- Set up an input source and output target to use. For futher instructions see Add Inputs to set up a sample input and Add Outputs to setup a sample output.

Set up a project

To create an analytics job use the Stream Analytics API for .NET, first set up your project.

1. Create a Visual Studio C# .NET console application.
2. In the Package Manager Console, run the following commands to install the NuGet packages. The first one is the Azure Stream Analytics Management .NET SDK. The second one is the Azure Active Directory client that will be used for authentication.

```
Install-Package Microsoft.Azure.Management.StreamAnalytics
Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory
```

3. Add the following **appSettings** section to the App.config file:

```
<appSettings>
  <!--CSM Prod related values-->
  <add key="ActiveDirectoryEndpoint" value="https://login.windows.net/" />
  <add key="ResourceManagerEndpoint" value="https://management.azure.com/" />
  <add key="WindowsManagementUri" value="https://management.core.windows.net/" />
  <add key="AsaClientId" value="1950a258-227b-4e31-a9cf-717495945fc2" />
  <add key="RedirectUri" value="urn:ietf:wg:oauth:2.0:oob" />
  <add key="SubscriptionId" value="YOUR AZURE SUBSCRIPTION" />
  <add key="ActiveDirectoryTenantId" value="YOU TENANT ID" />
</appSettings>
```

   Replace values for **SubscriptionId** and **ActiveDirectoryTenantId** with your Azure subscription and tenant IDs. You can get these values by running the following Azure PowerShell cmdlet:

```
Get-AzureAccount
```

4. Add the following **using** statements to the source file (Program.cs) in the project:

```
using System;
using System.Configuration;
using System.Threading;
using Microsoft.Azure;
using Microsoft.Azure.Management.StreamAnalytics;
using Microsoft.Azure.Management.StreamAnalytics.Models;
using Microsoft.IdentityModel.Clients.ActiveDirectory;
```

5. Add an authentication helper method:

```
public static string GetAuthorizationHeader()
{

    AuthenticationResult result = null;
    var thread = new Thread(() =>
    {
        try
        {
            var context = new AuthenticationContext(
                ConfigurationManager.AppSettings["ActiveDirectoryEndpoint"] +
                ConfigurationManager.AppSettings["ActiveDirectoryTenantId"]);

            result = context.AcquireToken(
                resource: ConfigurationManager.AppSettings["WindowsManagementUri"],
                clientId: ConfigurationManager.AppSettings["AsaClientId"],
                redirectUri: new Uri(ConfigurationManager.AppSettings["RedirectUri"]),
                promptBehavior: PromptBehavior.Always);
        }
        catch (Exception threadEx)
        {
            Console.WriteLine(threadEx.Message);
        }
    });

    thread.SetApartmentState(ApartmentState.STA);
    thread.Name = "AcquireTokenThread";
    thread.Start();
    thread.Join();

    if (result != null)
    {
        return result.AccessToken;
    }

    throw new InvalidOperationException("Failed to acquire token");
}
```

Create a Stream Analytics management client

A **StreamAnalyticsManagementClient** object allows you to manage the job and the job components, such as input, output, and transformation.

Add the following code to the beginning of the **Main** method:

```
string resourceGroupName = "<YOUR AZURE RESOURCE GROUP NAME>";
string streamAnalyticsJobName = "<YOUR STREAM ANALYTICS JOB NAME>";
string streamAnalyticsInputName = "<YOUR JOB INPUT NAME>";
string streamAnalyticsOutputName = "<YOUR JOB OUTPUT NAME>";
string streamAnalyticsTransformationName = "<YOUR JOB TRANSFORMATION NAME>";

// Get authentication token
TokenCloudCredentials aadTokenCredentials =
    new TokenCloudCredentials(
        ConfigurationManager.AppSettings["SubscriptionId"],
        GetAuthorizationHeader());

// Create Stream Analytics management client
StreamAnalyticsManagementClient client = new StreamAnalyticsManagementClient(aadTokenCredentials);
```

The **resourceGroupName** variable's value should be the same as the name of the resource group you created or picked in the prerequisite steps.

To automate the credential presentation aspect of job creation, refer to Authenticating a service principal with Azure

Resource Manager.

The remaining sections of this article assume that this code is at the beginning of the **Main** method.

Create a Stream Analytics job

The following code creates a Stream Analytics job under the resource group that you have defined. You will add an input, output, and transformation to the job later.

```
// Create a Stream Analytics job
JobCreateOrUpdateParameters jobCreateParameters = new JobCreateOrUpdateParameters()
{
    Job = new Job()
    {
        Name = streamAnalyticsJobName,
        Location = "<LOCATION>",
        Properties = new JobProperties()
        {
            EventsOutOfOrderPolicy = EventsOutOfOrderPolicy.Adjust,
            Sku = new Sku()
            {
                Name = "Standard"
            }
        }
    }
};

JobCreateOrUpdateResponse jobCreateResponse = client.StreamingJobs.CreateOrUpdate(resourceGroupName,
jobCreateParameters);
```

Create a Stream Analytics input source

The following code creates a Stream Analytics input source with the blob input source type and CSV serialization. To create an event hub input source, use **EventHubStreamInputDataSource** instead of **BlobStreamInputDataSource**. Similarly, you can customize the serialization type of the input source.

```
// Create a Stream Analytics input source
InputCreateOrUpdateParameters jobInputCreateParameters = new InputCreateOrUpdateParameters()
{
    Input = new Input()
    {
        Name = streamAnalyticsInputName,
        Properties = new StreamInputProperties()
        {
            Serialization = new CsvSerialization
            {
                Properties = new CsvSerializationProperties
                {
                    Encoding = "UTF8",
                    FieldDelimiter = ","
                }
            },
            DataSource = new BlobStreamInputDataSource
            {
                Properties = new BlobStreamInputDataSourceProperties
                {
                    StorageAccounts = new StorageAccount[]
                    {
                        new StorageAccount()
                        {
                            AccountName = "<YOUR STORAGE ACCOUNT NAME>",
                            AccountKey = "<YOUR STORAGE ACCOUNT KEY>"
                        }
                    },
                    Container = "samples",
                    PathPattern = ""
                }
            }
        }
    }
};

InputCreateOrUpdateResponse inputCreateResponse =
    client.Inputs.CreateOrUpdate(resourceGroupName, streamAnalyticsJobName, jobInputCreateParameters);
```

Input sources, whether from Blob storage or an event hub, are tied to a specific job. To use the same input source for different jobs, you must call the method again and specify a different job name.

Test a Stream Analytics input source

The **TestConnection** method tests whether the Stream Analytics job is able to connect to the input source as well as other aspects specific to the input source type. For example, in the blob input source you created in an earlier step, the method will check that the Storage account name and key pair can be used to connect to the Storage account as well as check that the specified container exists.

```
// Test input source connection
DataSourceTestConnectionResponse inputTestResponse =
    client.Inputs.TestConnection(resourceGroupName, streamAnalyticsJobName, streamAnalyticsInputName);
```

Create a Stream Analytics output target

Creating an output target is very similar to creating a Stream Analytics input source. Like input sources, output targets are tied to a specific job. To use the same output target for different jobs, you must call the method again and specify a different job name.

The following code creates an output target (Azure SQL database). You can customize the output target's data type

and/or serialization type.

```
// Create a Stream Analytics output target
OutputCreateOrUpdateParameters jobOutputCreateParameters = new OutputCreateOrUpdateParameters()
{
    Output = new Output()
    {
        Name = streamAnalyticsOutputName,
        Properties = new OutputProperties()
        {
            DataSource = new SqlAzureOutputDataSource()
            {
                Properties = new SqlAzureOutputDataSourceProperties()
                {
                    Server = "<YOUR DATABASE SERVER NAME>",
                    Database = "<YOUR DATABASE NAME>",
                    User = "<YOUR DATABASE LOGIN>",
                    Password = "<YOUR DATABASE LOGIN PASSWORD>",
                    Table = "<YOUR DATABASE TABLE NAME>"
                }
            }
        }
    }
};

OutputCreateOrUpdateResponse outputCreateResponse =
    client.Outputs.CreateOrUpdate(resourceGroupName, streamAnalyticsJobName, jobOutputCreateParameters);
```

Test a Stream Analytics output target

A Stream Analytics output target also has the **TestConnection** method for testing connections.

```
// Test output target connection
DataSourceTestConnectionResponse outputTestResponse =
    client.Outputs.TestConnection(resourceGroupName, streamAnalyticsJobName, streamAnalyticsOutputName);
```

Create a Stream Analytics transformation

The following code creates a Stream Analytics transformation with the query "select * from Input" and specifies to allocate one streaming unit for the Stream Analytics job. For more information on adjusting streaming units, see Scale Azure Stream Analytics jobs.

```
// Create a Stream Analytics transformation
TransformationCreateOrUpdateParameters transformationCreateParameters = new
TransformationCreateOrUpdateParameters()
{
    Transformation = new Transformation()
    {
        Name = streamAnalyticsTransformationName,
        Properties = new TransformationProperties()
        {
            StreamingUnits = 1,
            Query = "select * from Input"
        }
    }
};

var transformationCreateResp =
    client.Transformations.CreateOrUpdate(resourceGroupName, streamAnalyticsJobName,
transformationCreateParameters);
```

Like input and output, a transformations is also tied to the specific Stream Analytics job it was created under.

Start a Stream Analytics job

After creating a Stream Analytics job and its input(s), output(s), and transformation, you can start the job by calling the **Start** method.

The following sample code starts a Stream Analytics job with a custom output start time set to December 12, 2012, 12:12:12 UTC:

```
// Start a Stream Analytics job
JobStartParameters jobStartParameters = new JobStartParameters
{
    OutputStartMode = OutputStartMode.CustomTime,
    OutputStartTime = new DateTime(2012, 12, 12, 0, 0, 0, DateTimeKind.Utc)
};

LongRunningOperationResponse jobStartResponse = client.StreamingJobs.Start(resourceGroupName,
streamAnalyticsJobName, jobStartParameters);
```

Stop a Stream Analytics job

You can stop a running Stream Analytics job by calling the **Stop** method.

```
// Stop a Stream Analytics job
LongRunningOperationResponse jobStopResponse = client.StreamingJobs.Stop(resourceGroupName,
streamAnalyticsJobName);
```

Delete a Stream Analytics job

The **Delete** method will delete the job as well as the underlying sub-resources, including input(s), output(s), and transformation of the job.

```
// Delete a Stream Analytics job
LongRunningOperationResponse jobDeleteResponse = client.StreamingJobs.Delete(resourceGroupName,
streamAnalyticsJobName);
```

Get support

For further assistance, try our Azure Stream Analytics forum.

Next steps

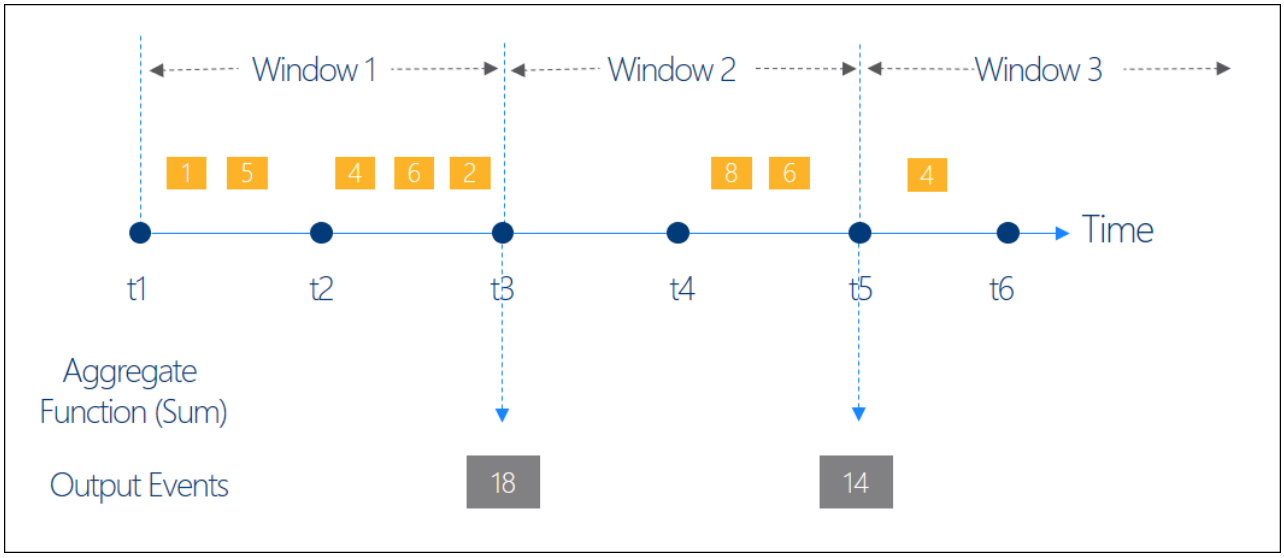
You've learning the basics of using a .NET SDK to create and run analytics jobs. To learn more, see the following:

- Introduction to Azure Stream Analytics
- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Management .NET SDK.
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Introduction to Stream Analytics Window functions

1/25/2017 • 1 min to read • Edit on GitHub

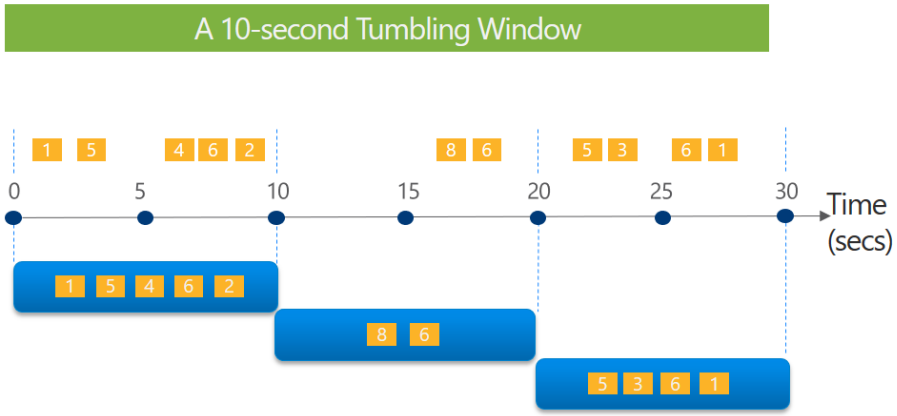
In many real time streaming scenarios, it is necessary to perform operations only on the data contained in temporal windows. Native support for windowing functions is a key feature of Azure Stream Analytics that moves the needle on developer productivity in authoring complex stream processing jobs. Stream Analytics enables developers to use **Tumbling**, **Hopping** and **Sliding** windows to perform temporal operations on streaming data. It is worth noting that all Window operations output results at the **end** of the window. The output of the window will be single event based on the aggregate function used. The event will have the time stamp of the end of the window and all Window functions are defined with a fixed length. Lastly it is important to note that all Window functions should be used in a **GROUP BY** clause.



Tumbling Window

Tumbling window functions are used to segment a data stream into distinct time segments and perform a function against them, such as the example below. The key differentiators of a Tumbling window are that they repeat, do not overlap and an event cannot belong to more than one tumbling window.

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

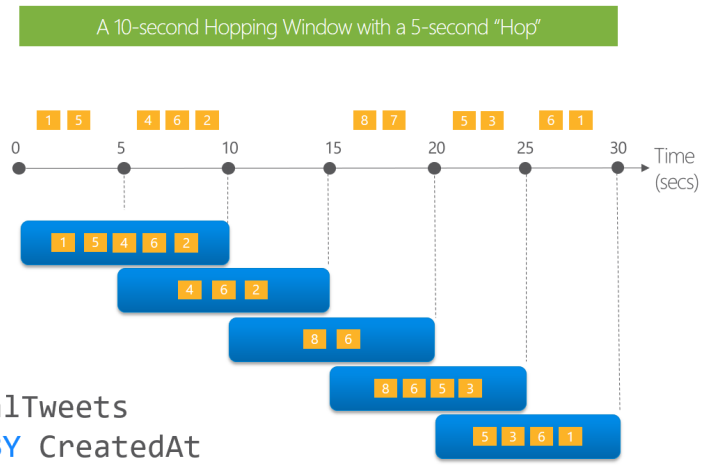



```
SELECT TimeZone, COUNT(*) AS Count
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY TimeZone, TumblingWindow(second,10)
```

Hopping Window

Hopping window functions hop forward in time by a fixed period. It may be easy to think of them as Tumbling windows that can overlap, so events can belong to more than one Hopping window result set. To make a Hopping window the same as a Tumbling window one would simply specify the hop size to be the same as the window size.

Every 5 seconds give me the count of tweets over the last 10 seconds

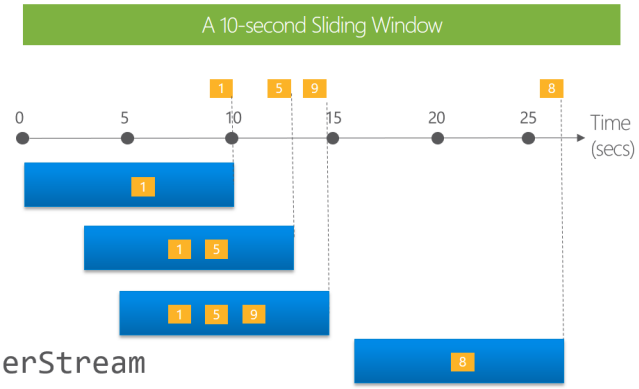



```
SELECT Topic, COUNT(*) AS TotalTweets
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY Topic, HoppingWindow(second, 10 , 5)
```

Sliding Window

Sliding window functions, unlike Tumbling or Hopping windows, produce an output **only** when an event occurs. Every window will have at least one event and the window continuously moves forward by an € (epsilon). Like Hopping Windows, events can belong to more than one Sliding Window.

Give me the count of tweets for all topics which are tweeted more than 10 times in the last 10 seconds




```
SELECT Topic, COUNT(*) FROM TwitterStream
TIMESTAMP BY CreatedAt
GROUP BY Topic, SlidingWindow(second, 10)
HAVING COUNT(*) > 10
```

Getting help with Window functions

For further assistance, try our Azure Stream Analytics forum

Next steps

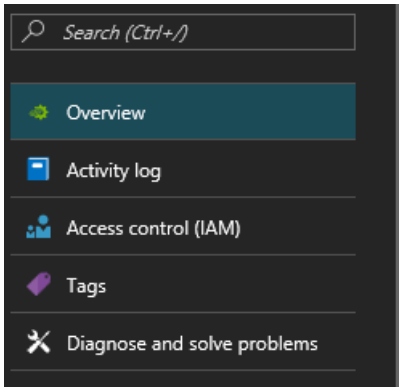
- Introduction to Azure Stream Analytics
- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Understand Stream Analytics job monitoring and how to monitor queries

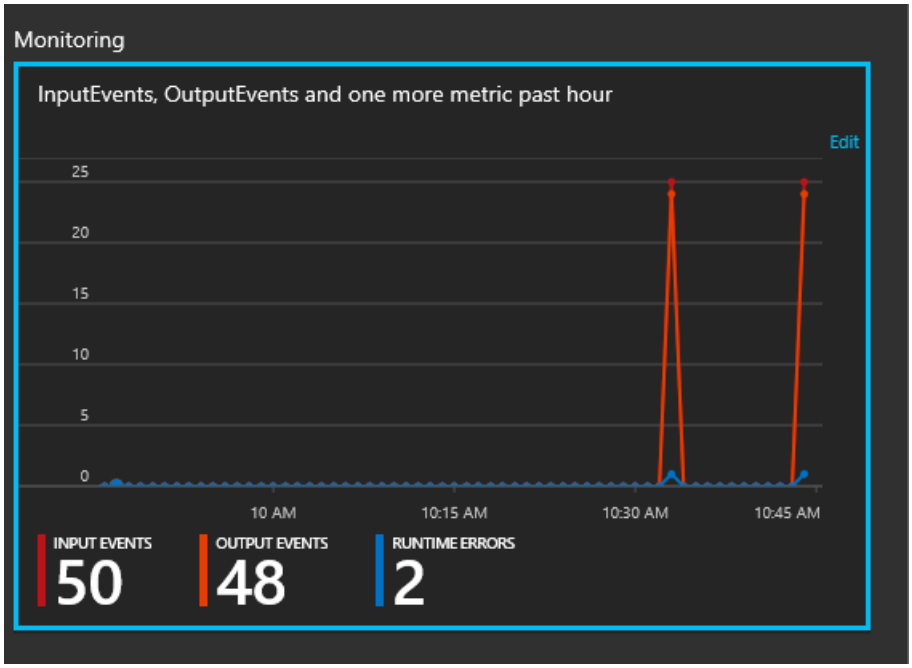
1/25/2017 • 2 min to read • Edit on GitHub

Introduction: The monitor page

The Azure portal both surface key performance metrics that can be used to monitor and troubleshoot your query and job performance. To see these metrics, browse to the Stream Analytics job you are interested in seeing metrics for and view the **Monitoring** section on the Overview page.



The window will appear as shown:



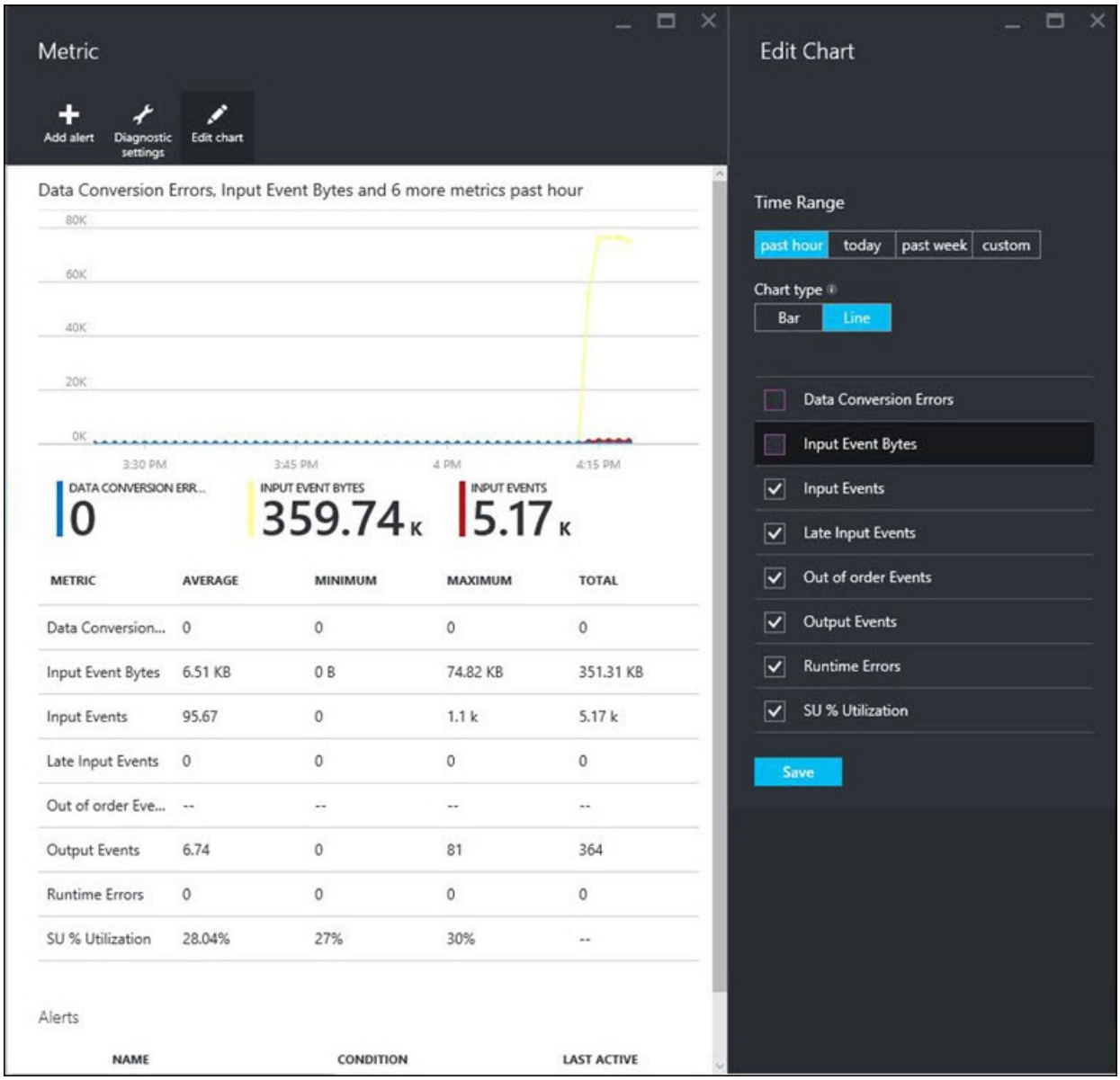
Metrics available for Stream Analytics

METRIC	DEFINITION
SU % Utilization	The utilization of the Streaming Unit(s) assigned to a job from the Scale tab of the job. Should this indicator reach 80%, or above, there is high probability that event processing may be delayed or stopped making progress.

METRIC	DEFINITION
Input Events	Amount of data received by the Stream Analytics job, in number of events. This can be used to validate that events are being sent to the input source.
Output Events	Amount of data sent by the Stream Analytics job to the output target, in number of events.
Out-of-Order Events	Number of events received out of order that were either dropped or given an adjusted timestamp, based on the Event Ordering Policy. This can be impacted by the configuration of the Out of Order Tolerance Window setting.
Data Conversion Errors	Number of data conversion errors incurred by a Stream Analytics job.
Runtime Errors	The total number of errors that happen during execution of a Stream Analytics job.
Late Input Events	Number of events arriving late from the source which have either been dropped or their timestamp has been adjusted, based on the Event Ordering Policy configuration of the Late Arrival Tolerance Window setting.
Function Requests	Number of calls to the Azure Machine Learning function (if present).
Failed Function Requests	Number of failed Azure Machine Learning function calls (if present).
Function Events	Number of events sent to the Azure Machine Learning function (if present).
Input Event Bytes	Amount of data received by the Stream Analytics job, in bytes. This can be used to validate that events are being sent to the input source.

Customizing Monitoring in the Azure portal

You can adjust the type of chart, metrics shown, and time range in the Edit Chart settings. For details, see How to Customize Monitoring.



Get help

For further assistance, try our Azure Stream Analytics forum

Next steps

- Introduction to Azure Stream Analytics
- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Job diagnostic logs

1/31/2017 • 3 min to read • Edit on GitHub

Introduction

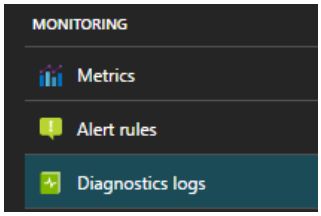
Stream Analytics exposes two types of logs:

- Activity logs that are always enabled and provide insights into operations performed on jobs;
- Diagnostic logs that are user configurable and provide richer insights into everything that happens with the job starting when it's created, updated, while it's running and until it's deleted;

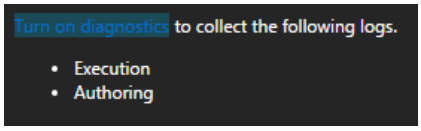
How to enable diagnostic logs

The diagnostics logs are turned **off** by default. To enable them follow these steps:

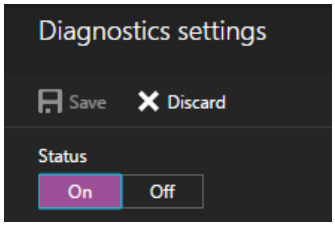
Sign on to the Azure portal and navigate to the streaming job blade and use the "Diagnostic logs" blade under "Monitoring".



Then click on the "Turn on diagnostics" link

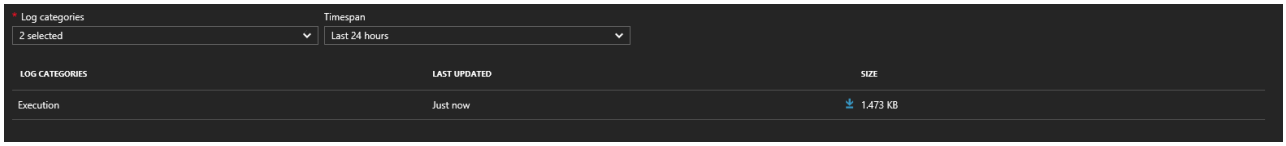


On the opened diagnostics, change the status to "On".



Configure the desired archival target (storage account, event hub, Log Analytics) and select the categories of logs that you want to collect (Execution, Authoring). Then save the new diagnostics configuration.

Once saved, the configuration will take about 10 minutes to take effect and after that logs will start appearing in the configured archival target which you can see on the "Diagnostics logs" blade:



More information about configuring diagnostics is available on the diagnostic logs page.

Diagnostic logs categories

There are two categories of diagnostic logs that we currently capture:

- **Authoring:** capture the logs related to job authoring operations: creation, adding and deleting inputs and outputs, adding and updating the query, starting and stopping the job.

- **Execution:** capture what is happening during job execution.
  - Connectivity errors;
  - Data processing errors;
    - Events that don't conform with the query definition (mismatched field types and values, missing fields etc);
    - Expression evaluation errors;
  - Etc.

Diagnostic logs schema

All logs are stored in JSON format and each entry has the following common string fields:

NAME	DESCRIPTION
time	The timestamp (in UTC) of the log
resourceId	The ID of the resource that operation took place on, upper-cased. It includes the subscription id, resource group and job name. For example, `/SUBSCRIPTIONS/6503D296-DAC1-4449-9B03-609A1F4A1C87/RESOURCEGROUPS RESOURCE-GROUP/PROVIDERS/MICROSOFT.STREAMANALYTICS/STREAMINGJOBS/MYSTREAMIN`
category	The log category, either `Execution` or `Authoring`
operationName	Name of the operation that is logged. For example, `Send Events: SQL Output write failure to mysqloutput`
status	The status of the operation. For example, `Failed, Succeeded`.
level	Log level. For example, `Error, Warning, Informational`
properties	log entry specific detail; serialized as JSON string; see below for more details

Execution logs properties schema

Execution logs contain information about events that happened during Stream Analytics job execution. Depending on the type of events, properties will have different schema. We currently have the following types:

Data errors

Any error in processing data will fall under this category of logs. Produced mainly during data read, serialization and write operation. It does not include connectivity errors which are treated as generic events.

NAME	DESCRIPTION
Source	Name of the job input or output where the error happened.
Message	Message associated with the error.
Type	The type of error. For example, `DataConversionError, CsvParserError, ServiceBusPropertyColumnMissingError` etc.
Data	Contains data useful to accurately locate the source of the error. Subject to truncation depending on size.

Depending on the **operationName** value data errors will have the following schema:

- **Serialize Events** - happening during event read operations when the data at the input does not satisfy the query

schema:

  - Type mismatch during event (de)serialize: field causing the error.
  - Cannot read an event, invalid serialization: information about the place in the input data where the error happened: blob name for blob input, offset and a sample of the data.

- **Send Events** - write operations: streaming event that caused the error.

Generic events

Everything else

NAME	DESCRIPTION
Error	(optional) Error information, usually exception information if available.
Message	Log message.
Type	Type of message, maps to internal categorization of errors: e.g. JobValidationError, BlobOutputAdapterInitializationFailure etc.
Correlation ID	GUID that uniquely identifies the job execution. All execution log entries produced since the job is started until it stops will have the same "Correlation ID".

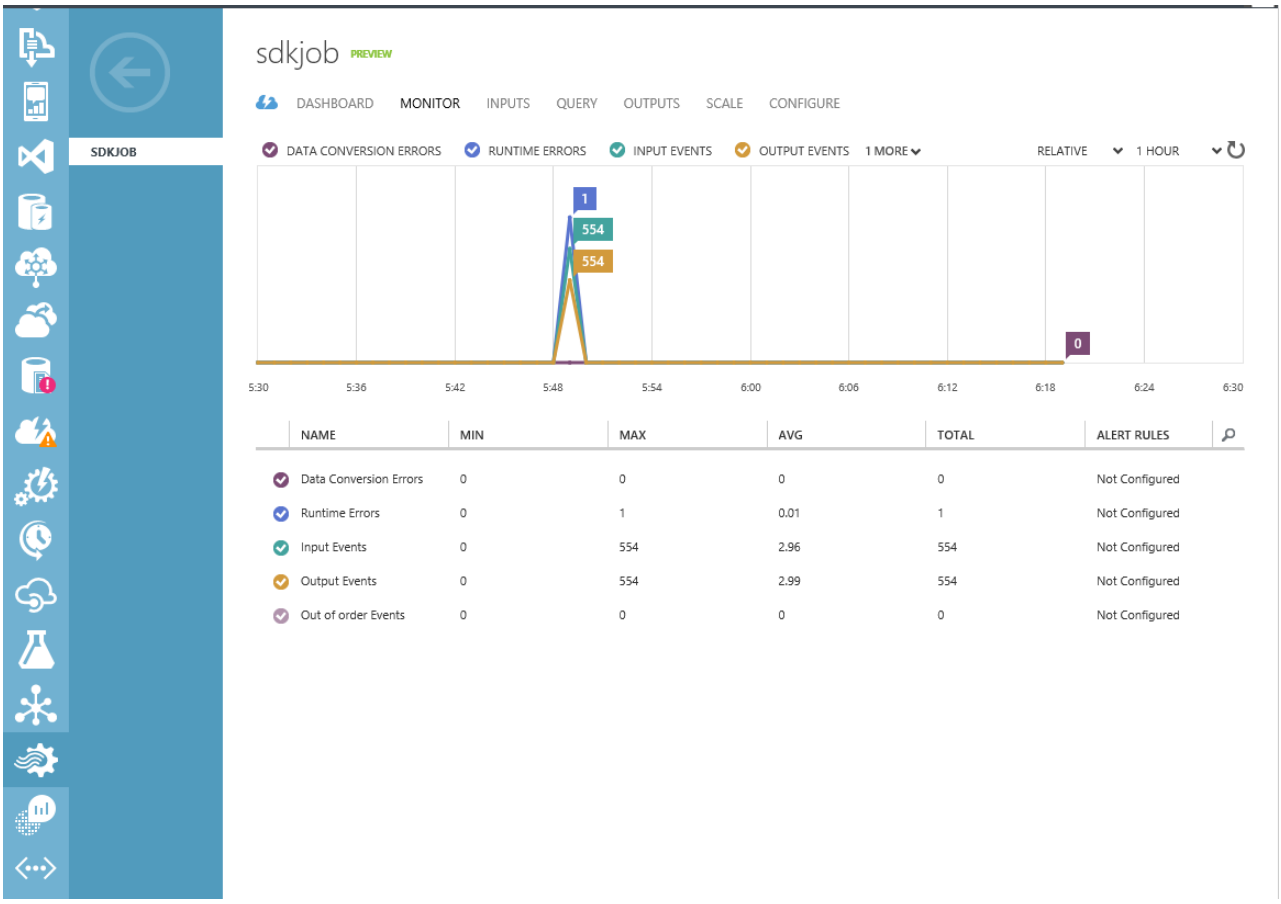
Next steps

- Introduction to Azure Stream Analytics
- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Programmatically create a Stream Analytics job monitor

1/25/2017 • 3 min to read • Edit on GitHub

This article demonstrates how to enable monitoring for a Stream Analytics job. Stream Analytics jobs created via REST APIs, Azure SDK, or Powershell do not have monitoring enabled by default. You can manually enable this in the Azure Portal by navigating to the job's Monitor page and clicking the Enable button or you can automate this process by following the steps in this article. The monitoring data will show up in the "Monitor" tab in the Azure Portal for your Stream Analytics job.



Prerequisites

Before you begin this article, you must have the following:

- Visual Studio 2012 or 2013.
- Download and install Azure .NET SDK.
- An existing Stream Analytics job that needs monitoring enabled.

Setup a project

1. Create a Visual Studio C# .Net console application.
2. In the Package Manager Console, run the following commands to install the NuGet packages. The first one is the Azure Stream Analytics Management .NET SDK. The second one is the Azure Monitor SDK which will be used to enable monitoring. The last one is the Azure Active Directory client that will be used for authentication.

```
Install-Package Microsoft.Azure.Management.StreamAnalytics
Install-Package Microsoft.Azure.Insights -Pre
Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory
```

3. Add the following appSettings section to the App.config file.

```
<appSettings>
  <!--CSM Prod related values-->
  <add key="ResourceGroupName" value="RESOURCE GROUP NAME" />
  <add key="JobName" value="YOUR JOB NAME" />
  <add key="StorageAccountName" value="YOUR STORAGE ACCOUNT"/>
  <add key="ActiveDirectoryEndpoint" value="https://login.windows.net/" />
  <add key="ResourceManagerEndpoint" value="https://management.azure.com/" />
  <add key="WindowsManagementUri" value="https://management.core.windows.net/" />
  <add key="AsaClientId" value="1950a258-227b-4e31-a9cf-717495945fc2" />
  <add key="RedirectUri" value="urn:ietf:wg:oauth:2.0:oob" />
  <add key="SubscriptionId" value="YOUR AZURE SUBSCRIPTION ID" />
  <add key="ActiveDirectoryTenantId" value="YOUR TENANT ID" />
</appSettings>
```

   Replace values for *SubscriptionId* and *ActiveDirectoryTenantId* with your Azure subscription and tenant IDs. You can get these values by running the following PowerShell cmdlet:

```
Get-AzureAccount
```

4. Add the following using statements to the source file (Program.cs) in the project.

```
  using System;
  using System.Configuration;
  using System.Threading;
  using Microsoft.Azure;
  using Microsoft.Azure.Management.Insights;
  using Microsoft.Azure.Management.Insights.Models;
  using Microsoft.Azure.Management.StreamAnalytics;
  using Microsoft.Azure.Management.StreamAnalytics.Models;
  using Microsoft.IdentityModel.Clients.ActiveDirectory;
```

5. Add an authentication helper method.

   public static string GetAuthorizationHeader()

```
    {
        AuthenticationResult result = null;
        var thread = new Thread(() =>
        {
            try
            {
                var context = new AuthenticationContext(
                    ConfigurationManager.AppSettings["ActiveDirectoryEndpoint"] +
                    ConfigurationManager.AppSettings["ActiveDirectoryTenantId"]);

                result = context.AcquireToken(
                    resource: ConfigurationManager.AppSettings["WindowsManagementUri"],
                    clientId: ConfigurationManager.AppSettings["AsaClientId"],
                    redirectUri: new Uri(ConfigurationManager.AppSettings["RedirectUri"]),
                    promptBehavior: PromptBehavior.Always);
            }
            catch (Exception threadEx)
            {
                Console.WriteLine(threadEx.Message);
            }
        });

        thread.SetApartmentState(ApartmentState.STA);
        thread.Name = "AcquireTokenThread";
        thread.Start();
        thread.Join();

        if (result != null)
        {
            return result.AccessToken;
        }

        throw new InvalidOperationException("Failed to acquire token");
```

}

Create Management Clients

The following code will set up the necessary variables and management clients.

```
string resourceGroupName = "<YOUR AZURE RESOURCE GROUP NAME>";
string streamAnalyticsJobName = "<YOUR STREAM ANALYTICS JOB NAME>";

// Get authentication token
TokenCloudCredentials aadTokenCredentials =
    new TokenCloudCredentials(
        ConfigurationManager.AppSettings["SubscriptionId"],
        GetAuthorizationHeader());

Uri resourceManagerUri = new
Uri(ConfigurationManager.AppSettings["ResourceManagerEndpoint"]);

// Create Stream Analytics and Insights management client
StreamAnalyticsManagementClient streamAnalyticsClient = new
StreamAnalyticsManagementClient(aadTokenCredentials, resourceManagerUri);
InsightsManagementClient insightsClient = new
InsightsManagementClient(aadTokenCredentials, resourceManagerUri);
```

Enable Monitoring For an Existing Stream Analytics Job

The following code will enable monitoring for an **existing** Stream Analytics job. The first part of the code performs a GET request against the Stream Analytics service to retrieve information about the particular Stream Analytics

job. It uses the "Id" property (retrieved from the GET request) as a parameter for the Put method in the second half of the code which sends a PUT request to the Insights service to enable monitoring for the Stream Analytics job.

WARNING

If you have previously enabled monitoring for a different Stream Analytics job, either through the Azure Portal or programmatically via the below code, **it is recommended that you provide the same storage account name that you did when you previously enabled monitoring.**

The storage account is linked to the region you created your Stream Analytics job in, not specifically to the job itself.

All Stream Analytics job (and all other Azure resources) in that same region share this storage account to store monitoring data. If you provide a different storage account, it may cause unintended side effects to the monitoring of your other Stream Analytics jobs and/or other Azure resources.

The storage account name used to replace `"<YOUR STORAGE ACCOUNT NAME>"` below should be a storage account that is in the same subscription as the Stream Analytics job you are enabling monitoring for.

```
// Get an existing Stream Analytics job
JobGetParameters jobGetParameters = new JobGetParameters()
{
    PropertiesToExpand = "inputs,transformation,outputs"
};
JobGetResponse jobGetResponse = streamAnalyticsClient.StreamingJobs.Get(resourceGroupName,
streamAnalyticsJobName, jobGetParameters);

// Enable monitoring
ServiceDiagnosticSettingsPutParameters insightPutParameters = new ServiceDiagnosticSettingsPutParameters()
{
        Properties = new ServiceDiagnosticSettings()
        {
            StorageAccountName = "<YOUR STORAGE ACCOUNT NAME>"
        }
};
insightsClient.ServiceDiagnosticSettingsOperations.Put(jobGetResponse.Job.Id, insightPutParameters);
```

Get support

For further assistance, try our Azure Stream Analytics forum.

Next steps

- Introduction to Azure Stream Analytics
- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Monitor and manage Stream Analytics jobs with Azure PowerShell cmdlets

1/25/2017 • 10 min to read • Edit on GitHub

Learn how to monitor and manage Stream Analytics resources with Azure PowerShell cmdlets and powershell scripting that execute basic Stream Analytics tasks.

Prerequisites for running Azure PowerShell cmdlets for Stream Analytics

- Create an Azure Resource Group in your subscription. The following is a sample Azure PowerShell script. For Azure PowerShell information, see Install and configure Azure PowerShell;

Azure PowerShell 0.9.8:

```
     # Log in to your Azure account
    Add-AzureAccount

    # Select the Azure subscription you want to use to create the resource group if you have more than one
subscription on your account.
    Select-AzureSubscription -SubscriptionName <subscription name>

    # If Stream Analytics has not been registered to the subscription, remove remark symbol below (#) to run
the Register-AzureProvider cmdlet to register the provider namespace.
    #Register-AzureProvider -Force -ProviderNamespace 'Microsoft.StreamAnalytics'

    # Create an Azure resource group
    New-AzureResourceGroup -Name <YOUR RESOURCE GROUP NAME> -Location <LOCATION>
```

Azure PowerShell 1.0:

```
     # Log in to your Azure account
    Login-AzureRmAccount

    # Select the Azure subscription you want to use to create the resource group.
    Get-AzureRmSubscription –SubscriptionName “your sub” | Select-AzureRmSubscription

    # If Stream Analytics has not been registered to the subscription, remove remark symbol below (#) to run
the Register-AzureProvider cmdlet to register the provider namespace.
    #Register-AzureRmResourceProvider -Force -ProviderNamespace 'Microsoft.StreamAnalytics'

    # Create an Azure resource group
    New-AzureRMResourceGroup -Name <YOUR RESOURCE GROUP NAME> -Location <LOCATION>
```

NOTE

Stream Analytics jobs created programmatically do not have monitoring enabled by default. You can manually enable monitoring in the Azure Portal by navigating to the job's Monitor page and clicking the Enable button or you can do this programmatically by following the steps located at Azure Stream Analytics - Monitor Stream Analytics Jobs Programatically.

Azure PowerShell cmdlets for Stream Analytics

The following Azure PowerShell cmdlets can be used to monitor and manage Azure Stream Analytics jobs. Note that Azure PowerShell has different versions. **In the examples listed the first command is for Azure PowerShell 0.9.8, the second command is for Azure PowerShell 1.0.** The Azure PowerShell 1.0 commands will always have "AzureRM" in the command.

Get-AzureStreamAnalyticsJob | Get-AzureRMStreamAnalyticsJob

Lists all Stream Analytics jobs defined in the Azure subscription or specified resource group, or gets job information about a specific job within a resource group.

Example 1

Azure PowerShell 0.9.8:

```
Get-AzureStreamAnalyticsJob
```

Azure PowerShell 1.0:

```
Get-AzureRMStreamAnalyticsJob
```

This PowerShell command returns information about all the Stream Analytics jobs in the Azure subscription.

Example 2

Azure PowerShell 0.9.8:

```
Get-AzureStreamAnalyticsJob -ResourceGroupName StreamAnalytics-Default-Central-US
```

Azure PowerShell 1.0:

```
Get-AzureRMStreamAnalyticsJob -ResourceGroupName StreamAnalytics-Default-Central-US
```

This PowerShell command returns information about all the Stream Analytics jobs in the resource group StreamAnalytics-Default-Central-US.

Example 3

Azure PowerShell 0.9.8:

```
Get-AzureStreamAnalyticsJob -ResourceGroupName StreamAnalytics-Default-Central-US -Name StreamingJob
```

Azure PowerShell 1.0:

```
Get-AzureRMStreamAnalyticsJob -ResourceGroupName StreamAnalytics-Default-Central-US -Name StreamingJob
```

This PowerShell command returns information about the Stream Analytics job StreamingJob in the resource group StreamAnalytics-Default-Central-US.

Get-AzureStreamAnalyticsInput | Get-AzureRMStreamAnalyticsInput

Lists all of the inputs that are defined in a specified Stream Analytics job, or gets information about a specific input.

Example 1

Azure PowerShell 0.9.8:

```
Get-AzureStreamAnalyticsInput -ResourceGroupName StreamAnalytics-Default-Central-US -JobName StreamingJob
```

Azure PowerShell 1.0:

```
Get-AzureRMStreamAnalyticsInput -ResourceGroupName StreamAnalytics-Default-Central-US -JobName StreamingJob
```

This PowerShell command returns information about all the inputs defined in the job StreamingJob.

Example 2

Azure PowerShell 0.9.8:

```
Get-AzureStreamAnalyticsInput -ResourceGroupName StreamAnalytics-Default-Central-US -JobName StreamingJob –Name
EntryStream
```

Azure PowerShell 1.0:

```
Get-AzureRMStreamAnalyticsInput -ResourceGroupName StreamAnalytics-Default-Central-US -JobName StreamingJob –
Name EntryStream
```

This PowerShell command returns information about the input named EntryStream defined in the job StreamingJob.

Get-AzureStreamAnalyticsOutput | Get-AzureRMStreamAnalyticsOutput

Lists all of the outputs that are defined in a specified Stream Analytics job, or gets information about a specific output.

Example 1

Azure PowerShell 0.9.8:

```
Get-AzureStreamAnalyticsOutput -ResourceGroupName StreamAnalytics-Default-Central-US -JobName StreamingJob
```

Azure PowerShell 1.0:

```
Get-AzureRMStreamAnalyticsOutput -ResourceGroupName StreamAnalytics-Default-Central-US -JobName StreamingJob
```

This PowerShell command returns information about the outputs defined in the job StreamingJob.

Example 2

Azure PowerShell 0.9.8:

```
Get-AzureStreamAnalyticsOutput -ResourceGroupName StreamAnalytics-Default-Central-US -JobName StreamingJob –
Name Output
```

Azure PowerShell 1.0:

```
Get-AzureRMStreamAnalyticsOutput -ResourceGroupName StreamAnalytics-Default-Central-US -JobName StreamingJob –
Name Output
```

This PowerShell command returns information about the output named Output defined in the job StreamingJob.

Get-AzureStreamAnalyticsQuota | Get-AzureRMStreamAnalyticsQuota

Gets information about the quota of streaming units in a specified region.

Example 1

Azure PowerShell 0.9.8:

```
Get-AzureStreamAnalyticsQuota –Location "Central US"
```

Azure PowerShell 1.0:

```
Get-AzureRMStreamAnalyticsQuota –Location "Central US"
```

This PowerShell command returns information about the quota and usage of streaming units in the Central US region.

Get-AzureStreamAnalyticsTransformation | GetAzureRMStreamAnalyticsTransformation

Gets information about a specific transformation defined in a Stream Analytics job.

Example 1

Azure PowerShell 0.9.8:

```
Get-AzureStreamAnalyticsTransformation -ResourceGroupName StreamAnalytics-Default-Central-US -JobName
StreamingJob –Name StreamingJob
```

Azure PowerShell 1.0:

```
Get-AzureRMStreamAnalyticsTransformation -ResourceGroupName StreamAnalytics-Default-Central-US -JobName
StreamingJob –Name StreamingJob
```

This PowerShell command returns information about the transformation called StreamingJob in the job StreamingJob.

New-AzureStreamAnalyticsInput | New-AzureRMStreamAnalyticsInput

Creates a new input within a Stream Analytics job, or updates an existing specified input.

The name of the input can be specified in the .json file or on the command line. If both are specified, the name on the command line must be the same as the one in the file.

If you specify an input that already exists and do not specify the –Force parameter, the cmdlet will ask whether or not to replace the existing input.

If you specify the –Force parameter and specify an existing input name, the input will be replaced without confirmation.

For detailed information on the JSON file structure and contents, refer to the Create Input (Azure Stream Analytics) section of the Stream Analytics Management REST API Reference Library.

Example 1

Azure PowerShell 0.9.8:

```
New-AzureStreamAnalyticsInput -ResourceGroupName StreamAnalytics-Default-Central-US -JobName StreamingJob –File
"C:\Input.json"
```

Azure PowerShell 1.0:

```
New-AzureRMStreamAnalyticsInput -ResourceGroupName StreamAnalytics-Default-Central-US -JobName StreamingJob –
File "C:\Input.json"
```

This PowerShell command creates a new input from the file Input.json. If an existing input with the name specified in the input definition file is already defined, the cmdlet will ask whether or not to replace it.

Example 2

Azure PowerShell 0.9.8:

```
New-AzureStreamAnalyticsInput -ResourceGroupName StreamAnalytics-Default-Central-US -JobName StreamingJob –File
"C:\Input.json" –Name EntryStream
```

Azure PowerShell 1.0:

```
New-AzureRMStreamAnalyticsInput -ResourceGroupName StreamAnalytics-Default-Central-US -JobName StreamingJob –
File "C:\Input.json" –Name EntryStream
```

This PowerShell command creates a new input in the job called EntryStream. If an existing input with this name is already defined, the cmdlet will ask whether or not to replace it.

Example 3

Azure PowerShell 0.9.8:

```
New-AzureStreamAnalyticsInput -ResourceGroupName StreamAnalytics-Default-Central-US -JobName StreamingJob –File
"C:\Input.json" –Name EntryStream -Force
```

Azure PowerShell 1.0:

```
New-AzureRMStreamAnalyticsInput -ResourceGroupName StreamAnalytics-Default-Central-US -JobName StreamingJob –
File "C:\Input.json" –Name EntryStream -Force
```

This PowerShell command replaces the definition of the existing input source called EntryStream with the definition from the file.

New-AzureStreamAnalyticsJob | New-AzureRMStreamAnalyticsJob

Creates a new Stream Analytics job in Microsoft Azure, or updates the definition of an existing specified job.

The name of the job can be specified in the .json file or on the command line. If both are specified, the name on the command line must be the same as the one in the file.

If you specify a job name that already exists and do not specify the –Force parameter, the cmdlet will ask whether or not to replace the existing job.

If you specify the –Force parameter and specify an existing job name, the job definition will be replaced without confirmation.

For detailed information on the JSON file structure and contents, refer to the Create Stream Analytics Job section of the Stream Analytics Management REST API Reference Library.

Example 1

Azure PowerShell 0.9.8:

```
New-AzureStreamAnalyticsJob -ResourceGroupName StreamAnalytics-Default-Central-US –File "C:\JobDefinition.json"
```

Azure PowerShell 1.0:

```
New-AzureRMStreamAnalyticsJob -ResourceGroupName StreamAnalytics-Default-Central-US –File
"C:\JobDefinition.json"
```

This PowerShell command creates a new job from the definition in JobDefinition.json. If an existing job with the name specified in the job definition file is already defined, the cmdlet will ask whether or not to replace it.

Example 2

Azure PowerShell 0.9.8:

```
New-AzureStreamAnalyticsJob -ResourceGroupName StreamAnalytics-Default-Central-US –File "C:\JobDefinition.json"
–Name StreamingJob -Force
```

Azure PowerShell 1.0:

```
New-AzureRMStreamAnalyticsJob -ResourceGroupName StreamAnalytics-Default-Central-US –File
"C:\JobDefinition.json" –Name StreamingJob -Force
```

This PowerShell command replaces the job definition for StreamingJob.

New-AzureStreamAnalyticsOutput | New-AzureRMStreamAnalyticsOutput

Creates a new output within a Stream Analytics job, or updates an existing output.

The name of the output can be specified in the .json file or on the command line. If both are specified, the name on the command line must be the same as the one in the file.

If you specify an output that already exists and do not specify the –Force parameter, the cmdlet will ask whether or not to replace the existing output.

If you specify the –Force parameter and specify an existing output name, the output will be replaced without confirmation.

For detailed information on the JSON file structure and contents, refer to the Create Output (Azure Stream Analytics) section of the Stream Analytics Management REST API Reference Library.

Example 1

Azure PowerShell 0.9.8:

```
New-AzureStreamAnalyticsOutput -ResourceGroupName StreamAnalytics-Default-Central-US –File "C:\Output.json" –
JobName StreamingJob –Name output
```

Azure PowerShell 1.0:

```
New-AzureRMStreamAnalyticsOutput -ResourceGroupName StreamAnalytics-Default-Central-US –File "C:\Output.json" –
JobName StreamingJob –Name output
```

This PowerShell command creates a new output called "output" in the job StreamingJob. If an existing output with this name is already defined, the cmdlet will ask whether or not to replace it.

Example 2

Azure PowerShell 0.9.8:

```
New-AzureStreamAnalyticsOutput -ResourceGroupName StreamAnalytics-Default-Central-US –File "C:\Output.json" –
JobName StreamingJob –Name output -Force
```

Azure PowerShell 1.0:

```
New-AzureRMStreamAnalyticsOutput -ResourceGroupName StreamAnalytics-Default-Central-US –File "C:\Output.json" –
JobName StreamingJob –Name output -Force
```

This PowerShell command replaces the definition for "output" in the job StreamingJob.

New-AzureStreamAnalyticsTransformation | New-AzureRMStreamAnalyticsTransformation

Creates a new transformation within a Stream Analytics job, or updates the existing transformation.

The name of the transformation can be specified in the .json file or on the command line. If both are specified, the name on the command line must be the same as the one in the file.

If you specify a transformation that already exists and do not specify the –Force parameter, the cmdlet will ask whether or not to replace the existing transformation.

If you specify the –Force parameter and specify an existing transformation name, the transformation will be replaced without confirmation.

For detailed information on the JSON file structure and contents, refer to the Create Transformation (Azure Stream Analytics) section of the Stream Analytics Management REST API Reference Library.

Example 1

Azure PowerShell 0.9.8:

```
New-AzureStreamAnalyticsTransformation -ResourceGroupName StreamAnalytics-Default-Central-US –File
"C:\Transformation.json" –JobName StreamingJob –Name StreamingJobTransform
```

Azure PowerShell 1.0:

```
New-AzureRMStreamAnalyticsTransformation -ResourceGroupName StreamAnalytics-Default-Central-US –File
"C:\Transformation.json" –JobName StreamingJob –Name StreamingJobTransform
```

This PowerShell command creates a new transformation called StreamingJobTransform in the job StreamingJob. If an existing transformation is already defined with this name, the cmdlet will ask whether or not to replace it.

Example 2

Azure PowerShell 0.9.8:

```
New-AzureStreamAnalyticsTransformation -ResourceGroupName StreamAnalytics-Default-Central-US –File
"C:\Transformation.json" –JobName StreamingJob –Name StreamingJobTransform -Force
```

Azure PowerShell 1.0:

```
New-AzureRMStreamAnalyticsTransformation -ResourceGroupName StreamAnalytics-Default-Central-US –File
"C:\Transformation.json" –JobName StreamingJob –Name StreamingJobTransform -Force
```

This PowerShell command replaces the definition of StreamingJobTransform in the job StreamingJob.

Remove-AzureStreamAnalyticsInput | Remove-AzureRMStreamAnalyticsInput

Asynchronously deletes a specific input from a Stream Analytics job in Microsoft Azure.
If you specify the –Force parameter, the input will be deleted without confirmation.

Example 1

Azure PowerShell 0.9.8:

```
Remove-AzureStreamAnalyticsInput -ResourceGroupName StreamAnalytics-Default-Central-US –JobName StreamingJob –
Name EventStream
```

Azure PowerShell 1.0:

```
Remove-AzureRMStreamAnalyticsInput -ResourceGroupName StreamAnalytics-Default-Central-US –JobName StreamingJob
–Name EventStream
```

This PowerShell command removes the input EventStream in the job StreamingJob.

Remove-AzureStreamAnalyticsJob | Remove-AzureRMStreamAnalyticsJob

Asynchronously deletes a specific Stream Analytics job in Microsoft Azure.
If you specify the –Force parameter, the job will be deleted without confirmation.

Example 1

Azure PowerShell 0.9.8:

```
Remove-AzureStreamAnalyticsJob -ResourceGroupName StreamAnalytics-Default-Central-US –Name StreamingJob
```

Azure PowerShell 1.0:

```
Remove-AzureRMStreamAnalyticsJob -ResourceGroupName StreamAnalytics-Default-Central-US –Name StreamingJob
```

This PowerShell command removes the job StreamingJob.

Remove-AzureStreamAnalyticsOutput | Remove-AzureRMStreamAnalyticsOutput

Asynchronously deletes a specific output from a Stream Analytics job in Microsoft Azure.
If you specify the –Force parameter, the output will be deleted without confirmation.

Example 1

Azure PowerShell 0.9.8:

```
Remove-AzureStreamAnalyticsOutput -ResourceGroupName StreamAnalytics-Default-Central-US –JobName StreamingJob –
Name Output
```

Azure PowerShell 1.0:

```
Remove-AzureRMStreamAnalyticsOutput -ResourceGroupName StreamAnalytics-Default-Central-US –JobName StreamingJob
–Name Output
```

This PowerShell command removes the output Output in the job StreamingJob.

Start-AzureStreamAnalyticsJob | Start-AzureRMStreamAnalyticsJob

Asynchronously deploys and starts a Stream Analytics job in Microsoft Azure.

Example 1

Azure PowerShell 0.9.8:

```
Start-AzureStreamAnalyticsJob -ResourceGroupName StreamAnalytics-Default-Central-US -Name StreamingJob -
OutputStartMode CustomTime -OutputStartTime 2012-12-12T12:12:12Z
```

Azure PowerShell 1.0:

```
Start-AzureRMStreamAnalyticsJob -ResourceGroupName StreamAnalytics-Default-Central-US -Name StreamingJob -
OutputStartMode CustomTime -OutputStartTime 2012-12-12T12:12:12Z
```

This PowerShell command starts the job StreamingJob with a custom output start time set to December 12, 2012, 12:12:12 UTC.

Stop-AzureStreamAnalyticsJob | Stop-AzureRMStreamAnalyticsJob

Asynchronously stops a Stream Analytics job from running in Microsoft Azure and de-allocates resources that were that were being used. The job definition and metadata will remain available within your subscription through both the Azure portal and management APIs, such that the job can be edited and restarted. You will not be charged for a job in the stopped state.

Example 1

Azure PowerShell 0.9.8:

```
Stop-AzureStreamAnalyticsJob -ResourceGroupName StreamAnalytics-Default-Central-US –Name StreamingJob
```

Azure PowerShell 1.0:

```
Stop-AzureRMStreamAnalyticsJob -ResourceGroupName StreamAnalytics-Default-Central-US –Name StreamingJob
```

This PowerShell command stops the job StreamingJob.

Test-AzureStreamAnalyticsInput | Test-AzureRMStreamAnalyticsInput

Tests the ability of Stream Analytics to connect to a specified input.

Example 1

Azure PowerShell 0.9.8:

```
Test-AzureStreamAnalyticsInput -ResourceGroupName StreamAnalytics-Default-Central-US –JobName StreamingJob –
Name EntryStream
```

Azure PowerShell 1.0:

```
Test-AzureRMStreamAnalyticsInput -ResourceGroupName StreamAnalytics-Default-Central-US –JobName StreamingJob –
Name EntryStream
```

This PowerShell command tests the connection status of the input EntryStream in StreamingJob.

Test-AzureStreamAnalyticsOutput | Test-AzureRMStreamAnalyticsOutput

Tests the ability of Stream Analytics to connect to a specified output.

Example 1

Azure PowerShell 0.9.8:

```
Test-AzureStreamAnalyticsOutput -ResourceGroupName StreamAnalytics-Default-Central-US –JobName StreamingJob –
Name Output
```

Azure PowerShell 1.0:

```
Test-AzureRMStreamAnalyticsOutput -ResourceGroupName StreamAnalytics-Default-Central-US –JobName StreamingJob –
Name Output
```

This PowerShell command tests the connection status of the output Output in StreamingJob.

Get support

For further assistance, try our Azure Stream Analytics forum.

Next steps

- Introduction to Azure Stream Analytics
- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Scale Azure Stream Analytics jobs to increase stream data processing throughput

1/25/2017 • 11 min to read • Edit on GitHub

Learn how to tune analytics jobs and calculate *streaming units* for Stream Analytics, how to scale Stream Analytics jobs by configuring input partitions, tuning the analytics query definition and setting job streaming units.

What are the parts of a Stream Analytics job?

A Stream Analytics job definition includes inputs, a query, and output. Inputs are from where the job reads the data stream, the query is used to transform the data input stream, and the output is where the job sends the job results to.

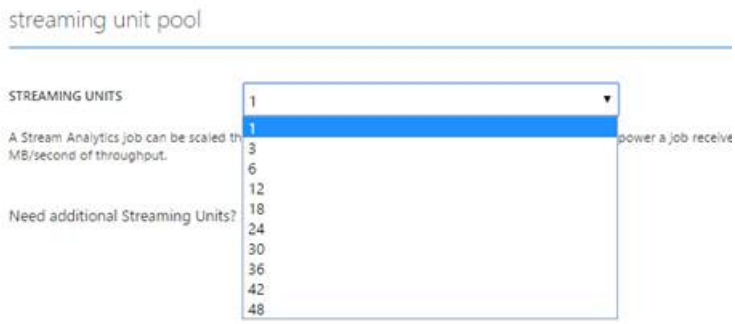
A job requires at least one input source for data streaming. The data stream input source can be stored in an Azure Service Bus Event Hub or in Azure Blob storage. For more information, see Introduction to Azure Stream Analytics and Get started using Azure Stream Analytics.

Configuring streaming units

Streaming units (SUs) represent the resources and computing power required to execute an Azure Stream Analytics job. SUs provide a way to describe the relative event processing capacity based on a blended measure of CPU, memory, and read and write rates. Each streaming unit corresponds to roughly 1MB/second of throughput.

Choosing how many SUs are required for a particular job depends on the partition configuration for the inputs and the query defined for the job. You can select up to your quota in streaming units for a job by using the Azure Classic Portal. Each Azure subscription by default has a quota of up to 50 streaming units for all the analytics jobs in a specific region. To increase streaming units for your subscriptions, contact Microsoft Support.

The number of streaming units that a job can utilize depends on the partition configuration for the inputs and the query defined for the job. Note also that a valid value for the stream units must be used. The valid values start at 1, 3, 6 and then upwards in increments of 6, as shown below.



This article will show you how to calculate and tune the query to increase throughput for analytics jobs.

Embarrassingly parallel job

The embarrassingly parallel job is the most scalable scenario we have in Azure Stream Analytics. It connects one partition of the input to one instance of the query to one partition of the output. Achieving this parallelism requires a few things:

1. If your query logic is dependent on the same key being processed by the same query instance, then you must ensure that the events go to the same partition of your input. In the case of Event Hubs, this means that the Event Data needs to have **PartitionKey** set or you can use partitioned senders. For Blob, this means that the events are sent to the same partition folder. If your query logic does not require the same key be processed by the same query instance, then you can ignore this requirement. An example of this would be a simple select/project/filter query.
2. Once the data is laid out like it needs to be on the input side, we need to ensure that your query is partitioned. This requires you to use **Partition By** in all of the steps. Multiple steps are allowed but they all must be partitioned by the same key. Another thing to note is that, currently, the partitioning key needs to be set to **PartitionId** to have a fully parallel job.
3. Currently only Event Hubs and Blob support partitioned output. For Event Hubs output, you need to configure the **PartitionKey** field to be **PartitionId**. For Blob, you don't have to do anything.
4. Another thing to note, the number of input partitions must equal the number of output partitions. Blob output doesn't currently support partitions, but this is okay because it will inherit the partitioning scheme of the upstream query. Examples of partition values that would allow a fully parallel job:
   a. 8 Event Hubs input partitions and 8 Event Hubs output partitions
   b. 8 Event Hubs input partitions and Blob Output
   c. 8 Blob input partitions and Blob Output
   d. 8 Blob input partitions and 8 Event Hubs output partitions

Here are some example scenarios that are embarrassingly parallel.

Simple query

Input – Event Hubs with 8 partitions Output – Event Hub with 8 partitions

Query:

```
SELECT TollBoothId
FROM Input1 Partition By PartitionId
WHERE TollBoothId > 100
```

This query is a simple filter and as such, we do not need to worry about partitioning the input we send to Event Hubs. You will notice that the query has **Partition By** of **PartitionId**, so we fulfill requirement #2 from above. For the output, we need to configure the Event Hubs output in the job to have the **PartitionKey** field set to **PartitionId**. One last check, input partitions == output partitions. This topology is embarrassingly parallel.

Query with grouping key

Input – Event Hubs with 8 partitions Output – Blob

Query:

```
SELECT COUNT(*) AS Count, TollBoothId
FROM Input1 Partition By PartitionId
GROUP BY TumblingWindow(minute, 3), TollBoothId, PartitionId
```

This query has a grouping key and as such, the same key needs to be processed by the same query instance. This means we need to send our events to Events Hubs in a partitioned manner. Which key do we care about? **PartitionId** is a job logic concept, the real key we care about is **TollBoothId**. This means we should set the

PartitionKey of the event data we send to Event Hubs to be the **TollBoothId** of the event. The query has **Partition By** of **PartitionId**, so we are good there. For the output, since it is Blob, we do not need to worry about configuring **PartitionKey**. For requirement #4, again, this is Blob, so we don't need to worry about it. This topology is embarrassingly parallel.

Multi Step Query with Grouping Key

Input – Event Hub with 8 partitions Output – Event Hub with 8 partitions

Query:

```
WITH Step1 AS (
SELECT COUNT(*) AS Count, TollBoothId, PartitionId
FROM Input1 Partition By PartitionId
GROUP BY TumblingWindow(minute, 3), TollBoothId, PartitionId
)

SELECT SUM(Count) AS Count, TollBoothId
FROM Step1 Partition By PartitionId
GROUP BY TumblingWindow(minute, 3), TollBoothId, PartitionId
```

This query has a grouping key and as such, the same key needs to be processed by the same query instance. We can use the same strategy as the previous query. The query has multiple steps. Does each step have **Partition By** of ** PartitionId**? Yes, so we are good. For the output, we need to set the **PartitionKey** to **PartitionId** like discussed above and we can also see it has the same number of partitions as the input. This topology is embarrassingly parallel.

Example scenarios that are NOT embarrassingly parallel

Mismatched Partition Count

Input – Event Hubs with 8 partitions Output – Event Hub with 32 partitions

It doesn't matter what the query is in this case because the input partition count != output partition count.

Not using Event Hubs or Blobs as output

Input – Event Hubs with 8 partitions Output – PowerBI

PowerBI output doesn't currently support partitioning.

Multi Step Query with different Partition By values

Input – Event Hub with 8 partitions Output – Event Hub with 8 partitions

Query:

```
WITH Step1 AS (
SELECT COUNT(*) AS Count, TollBoothId, PartitionId
FROM Input1 Partition By PartitionId
GROUP BY TumblingWindow(minute, 3), TollBoothId, PartitionId
)

SELECT SUM(Count) AS Count, TollBoothId
FROM Step1 Partition By TollBoothId
GROUP BY TumblingWindow(minute, 3), TollBoothId
```

As you can see, the second step uses **TollBoothId** as the partitioning key. This is not the same as the first step and will therefore require us to do a shuffle.

These are some examples and counterexamples of Stream Analytics jobs that will be able to achieve an embarrassingly parallel topology and with it the potential for maximum scale. For jobs that do not fit one of

these profiles, future updates detailing how to maximally scale some of the other canonical Stream Analytics scenarios will be made.

For now make use of the general guidance below:

Calculate the maximum streaming units of a job

The total number of streaming units that can be used by a Stream Analytics job depends on the number of steps in the query defined for the job and the number of partitions for each step.

Steps in a query

A query can have one or many steps. Each step is a sub-query defined by using the **WITH** keyword. The only query that is outside of the **WITH** keyword is also counted as a step, for example, the **SELECT** statement in the following query:

```
WITH Step1 AS (
    SELECT COUNT(*) AS Count, TollBoothId
    FROM Input1 Partition By PartitionId
    GROUP BY TumblingWindow(minute, 3), TollBoothId, PartitionId
)

SELECT SUM(Count) AS Count, TollBoothId
FROM Step1
GROUP BY TumblingWindow(minute,3), TollBoothId
```

The previous query has two steps.

> **NOTE**
> This sample query will be explained later in the article.

Partition a step

Partitioning a step requires the following conditions:

- The input source must be partitioned. For more information see the Event Hubs Programming Guide.
- The **SELECT** statement of the query must read from a partitioned input source.
- The query within the step must have the **Partition By** keyword

When a query is partitioned, the input events will be processed and aggregated in separate partition groups, and outputs events are generated for each of the groups. If a combined aggregate is desirable, you must create a second non-partitioned step to aggregate.

Calculate the max streaming units for a job

All non-partitioned steps together can scale up to six streaming units for a Stream Analytics job. To add additional streaming units, a step must be partitioned. Each partition can have six streaming units.

QUERY OF A JOB	MAX STREAMING UNITS FOR THE JOB
• The query contains one step. • The step is not partitioned.	6
• The input data stream is partitioned by 3. • The query contains one step. • The step is partitioned.	18

• The query contains two steps. • Neither of the steps is partitioned.	6
• The data stream input is partitioned by 3. • The query contains two steps. The input step is partitioned and the second step is not. • The SELECT statement reads from the partitioned input.	24 (18 for partitioned steps + 6 for non-partitioned steps)

Examples of scale

The following query calculates the number of cars going through a toll station with three tollbooths within a three-minute window. This query can be scaled up to six streaming units.

```
SELECT COUNT(*) AS Count, TollBoothId
FROM Input1
GROUP BY TumblingWindow(minute, 3), TollBoothId, PartitionId
```

To use more streaming units for the query, both the data stream input and the query must be partitioned. Given that the data stream partition is set to 3, the following modified query can be scaled up to 18 streaming units:

```
SELECT COUNT(*) AS Count, TollBoothId
FROM Input1 Partition By PartitionId
GROUP BY TumblingWindow(minute, 3), TollBoothId, PartitionId
```

When a query is partitioned, the input events are processed and aggregated in separate partition groups. Output events are also generated for each of the groups. Partitioning can cause some unexpected results when the **Group-by** field is not the Partition Key in the data stream input. For example, the **TollBoothId** field in the previous sample query is not the Partition Key of Input1. The data from the TollBooth #1 can be spread in multiple partitions.

Each of the Input1 partitions will be processed separately by Stream Analytics, and multiple records of the car-pass-through count for the same tollbooth in the same tumbling window will be created. If the input partition key can't be changed, this problem can be fixed by adding an additional non-partition step, for example:

```
WITH Step1 AS (
    SELECT COUNT(*) AS Count, TollBoothId
    FROM Input1 Partition By PartitionId
    GROUP BY TumblingWindow(minute, 3), TollBoothId, PartitionId
)

SELECT SUM(Count) AS Count, TollBoothId
FROM Step1
GROUP BY TumblingWindow(minute, 3), TollBoothId
```

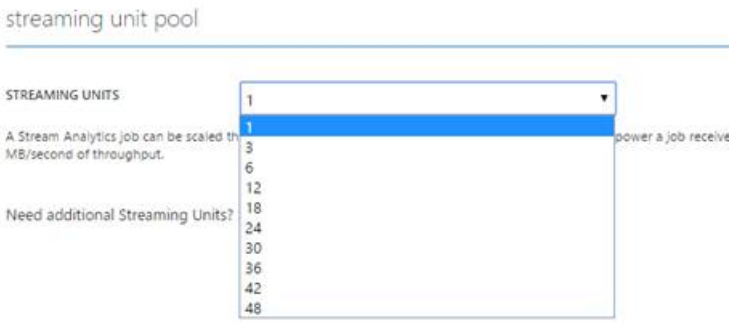
This query can be scaled to 24 streaming units.

> **NOTE**
>
> If you are joining two streams, ensure that the streams are partitioned by the partition key of the column that you do the joins, and you have the same number of partitions in both streams.

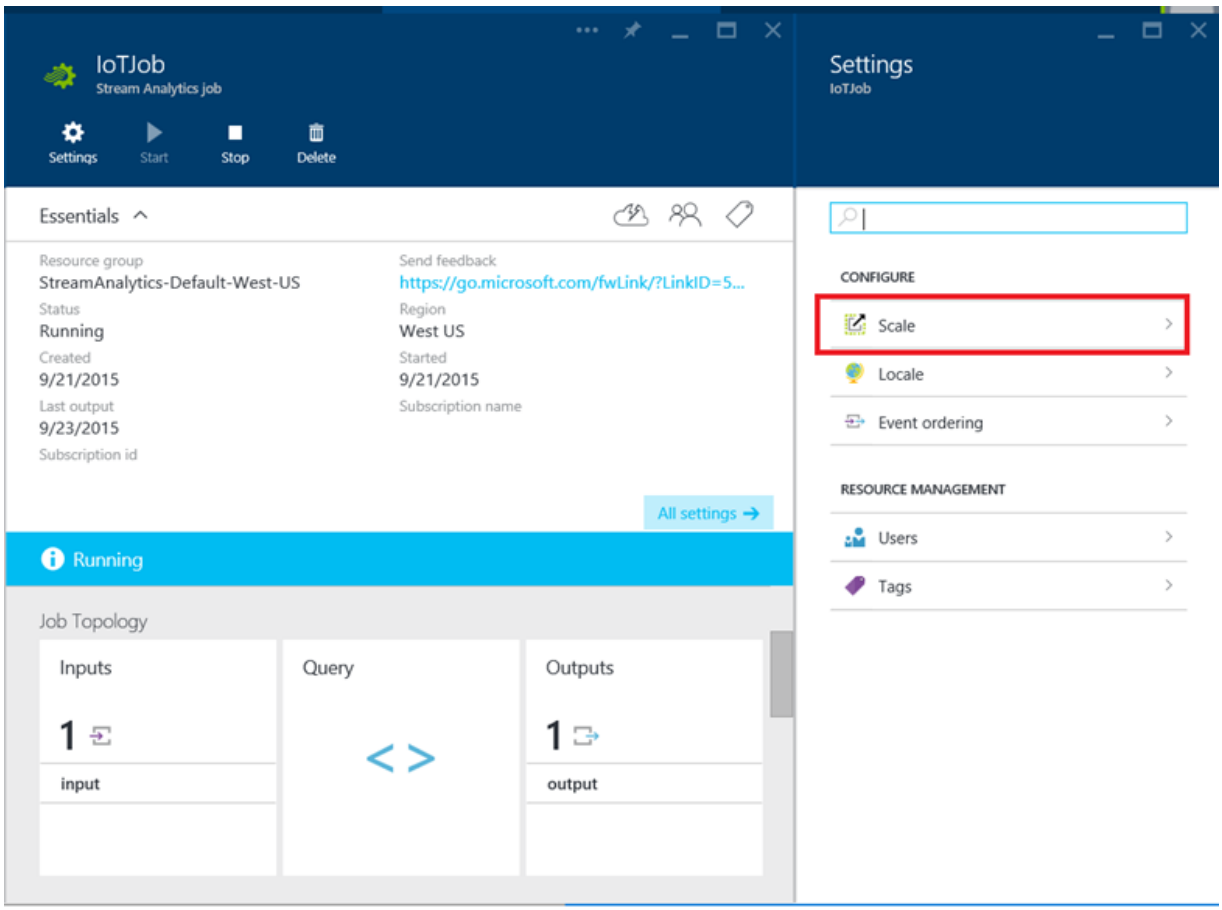
Configure Stream Analytics job partition

To adjust a streaming unit for a job

1. Sign in to the Management portal.
2. Click **Stream Analytics** in the left pane.
3. Click the Stream Analytics job that you want to scale.
4. Click **SCALE** at the top of the page.



In the Azure Portal, Scale settings can be accessed under Settings:



Monitor job performance

Using the management portal, you can track the throughput of a job in Events/second:

Calculate the expected throughput of the workload in Events/second. If the throughput is less than expected, tune the input partition, tune the query, and add additional streaming units to your job.

Stream Analytics Throughput at scale - Raspberry Pi scenario

To understand how stream analytics jobs scale in a typical scenario in terms of processing throughput across multiple Streaming Units, here is an experiment that sends sensor data (clients) into Event Hub, processes it and sends alert or statistics as an output to another Event Hub.

The client is sending synthesized sensor data to Event Hubs in JSON format to Stream Analytics and the data output is also in JSON format. Here is how the sample data would look like–

```
{"devicetime":"2014-12-
11T02:24:56.8850110Z","hmdt":42.7,"temp":72.6,"prss":98187.75,"lght":0.38,"dspl":"R-PI Olivier's Office"}
```

Query: "Send an alert when the light is switched off"

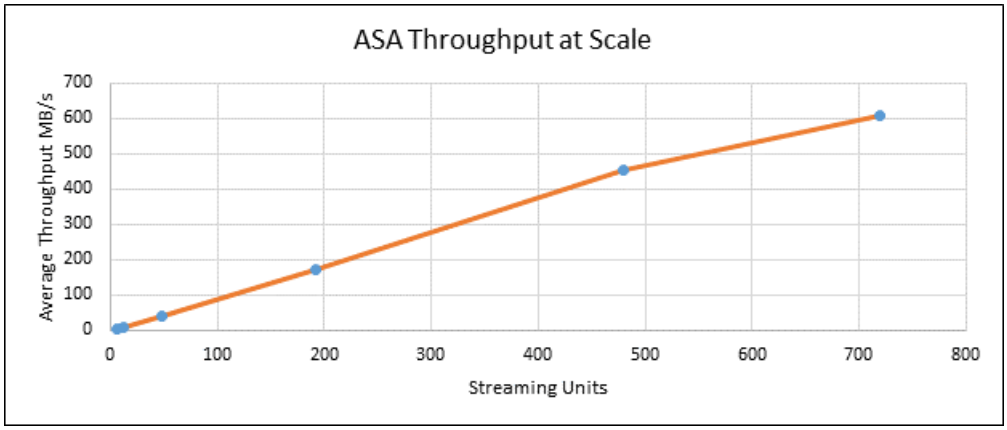
```
SELECT AVG(lght),
 "LightOff" as AlertText
FROM input TIMESTAMP
BY devicetime
 WHERE
    lght< 0.05 GROUP BY TumblingWindow(second, 1)
```

Measuring Throughput: Throughput in this context is the amount of input data processed by Stream Analytics in a fixed amount of time (10 minutes). To achieve best processing throughput for the input data, both the data stream input and the query must be partitioned. Also **COUNT()**is included in the query to measure how many input events were processed. To ensure the job is not simply waiting for input events to come, each partition of the Input Event Hub was preloaded with sufficient input data (about 300MB).

Below are the results with increasing number of Streaming units and corresponding Partition counts in Event Hubs.

INPUT PARTITIONS	OUTPUT PARTITIONS	STREAMING UNITS	SUSTAINED THROUGHPUT

12	12	6	4.06 MB/s
12	12	12	8.06 MB/s
48	48	48	38.32 MB/s
192	192	192	172.67 MB/s
480	480	480	454.27 MB/s
720	720	720	609.69 MB/s



Get help

For further assistance, try our Azure Stream Analytics forum.

Next steps

- Introduction to Azure Stream Analytics
- Get started using Azure Stream Analytics
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Scale your Stream Analytics job with Azure Machine Learning functions

1/23/2017 • 8 min to read • Edit on GitHub

It is often quite easy to set up an Stream Analytics job and run some sample data through it. What should we do when we need to run the same job with higher data volume? It requires us to understand how to configure the Stream Analytics job so that it will scale. In this document, we will focus on the special aspects of scaling Stream Analytics jobs with Machine Learning functions. For information on how to scale Stream Analytics jobs in general see the article Scaling jobs.

What is an Azure Machine Learning function in Stream Analytics?

A Machine Learning function in Stream Analytics can be used like a regular function call in the Stream Analytics query language. However, behind the scene, the function calls are actually Azure Machine Learning Web Service requests. Machine Learning web services support "batching" multiple rows, which is called mini-batch, in the same web service API call, to improve overall throughput. Please see the following articles for more details; Azure Machine Learning functions in Stream Analytics and Azure Machine Learning Web Services.

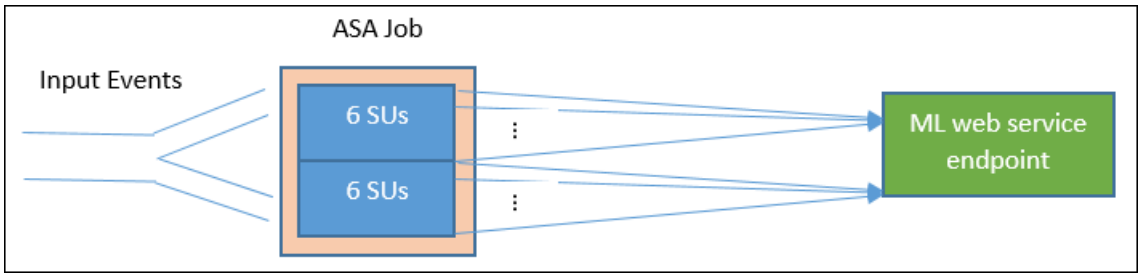
Configure a Stream Analytics job with Machine Learning functions

When configuring a Machine Learning function for Stream Analytics job, there are two parameters to consider, the batch size of the Machine Learning function calls, and the Streaming Units (SUs) provisioned for the Stream Analytics job. To determine the appropriate values for these, first a decision must be made between latency and throughput, that is, latency of the Stream Analytics job, and throughput of each SU. SUs may always be added to a job to increase throughput of a well partitioned Stream Analytics query, although additional SUs increases the cost of running the job.

Therefore it is important to determine the *tolerance* of latency in running a Stream Analytics job. Additional latency from running Azure Machine Learning service requests will naturally increase with batch size, which will compound the latency of the Stream Analytics job. On the other hand, increasing batch size allows the Stream Analytics job to process *more events with the *same number* of Machine Learning web service requests. Often the increase of Machine Learning web service latency is sub-linear to the increase of batch size so it is important to consider the most cost-efficient batch size for a Machine Learning web service in any given situation. The default batch size for the web service requests is 1000 and may be modified either by using the Stream Analytics REST API or the PowerShell client for Stream Analytics.

Once a batch size has been determined, the amount of Streaming Units (SUs) can be determined, based on the number of events that the function needs to process per second. For further information on Streaming Units consult the article Stream Analytics Scale jobs.

In general, there are 20 concurrent connections to the Machine Learning web service for every 6 SUs, except that 1 SU jobs and 3 SU jobs will get 20 concurrent connections also. For example, if the input data rate is 200,000 events per second and the batch size is left to the default of 1000 the resulting web service latency with 1000 events mini-batch is 200ms. This means every connection can make 5 requests to the Machine Learning web service in a second. With 20 connections, the Stream Analytics job can process 20,000 events in 200ms and therefore 100,000 events in a second. So to process 200,000 events per second, the Stream Analytics job needs 40 concurrent connections, which comes out to 12 SUs. The diagram below illustrates the requests from the Stream Analytics job to the Machine Learning web service endpoint – Every 6 SUs has 20 concurrent connections to Machine Learning web service at max.



In general, ***B*** for batch size, ***L*** for the web service latency at batch size B in milliseconds, the throughput of an Stream Analytics job with ***N*** SUs is:

$$20 * \text{ceil}(N/6) * B * (1000/L)$$

An additional consideration may be the 'max concurrent calls' on the Machine Learning web service side, it's recommended to set this to the maximum value (200 currently).

For more information on this setting please review the Scaling article for Machine Learning Web Services.

Example – Sentiment Analysis

The following example includes a Stream Analytics job with the sentiment analysis Machine Learning function, as described in the Stream Analytics Machine Learning integration tutorial.

The query is a simple fully partitioned query followed by the **sentiment** function, as shown below:

```
WITH subquery AS (
    SELECT text, sentiment(text) as result from input
)

Select text, result.[Score]
Into output
From subquery
```

Consider the following scenario; with a throughput of 10,000 tweets per second a Stream Analytics job must be created to perform sentiment analysis of the tweets (events). Using 1 SU, could this Stream Analytics job be able to handle the traffic? Using the default batch size of 1000 the job should be able to keep up with the input. Further the added Machine Learning function should generate no more than a second of latency, which is the general default latency of the sentiment analysis Machine Learning web service (with a default batch size of 1000). The Stream Analytics job's **overall** or end-to-end latency would typically be a few seconds. Take a more detailed look into this Stream Analytics job, *especially* the Machine Learning function calls. Having the batch size as 1000, a throughput of 10,000 events will take about 10 requests to web service. Even with 1 SU, there are enough concurrent connections to accommodate this input traffic.

But what if the input event rate increases by 100x and now the Stream Analytics job needs to process 1,000,000 tweets per second? There are two options:

1. Increase the batch size, or
2. Partition the input stream to process the events in parallel

With the first option, the job **latency** will increase.

With the second option, more SUs would need to be provisioned and therefore generate more concurrent Machine Learning web service requests. This means the job **cost** will increase.

Assume the latency of the sentiment analysis Machine Learning web service is 200ms for 1000-event batches or below, 250ms for 5,000-event batches, 300ms for 10,000-event batches or 500ms for 25,000-event batches.

1. Using the first option, (**not** provisioning more SUs), the batch size could be increased to **25,000**. This in turn would allow the job to process 1,000,000 events with 20 concurrent connections to the Machine Learning web service (with a latency of 500ms per call). So the additional latency of the Stream Analytics job due to the sentiment function requests against the Machine Learning web service requests would be increased from **200ms** to **500ms**. However, note that batch size **cannot** be increased infinitely as the Machine Learning web services requires the payload size of a request be 4MB or smaller web service requests timeout after 100 seconds of operation.
2. Using the second option, the batch size is left at 1000, with 200ms web service latency, every 20 concurrent connections to the web service would be able to process 1000 * 20 * 5 events = 100,000 per second. So to process 1,000,000 events per second, the job would need 60 SUs. Compared to the first option, Stream Analytics job would make more web service batch requests, in turn generating an increased cost.

Below is a table for the throughput of the Stream Analytics job for different SUs and batch sizes (in number of events per second).

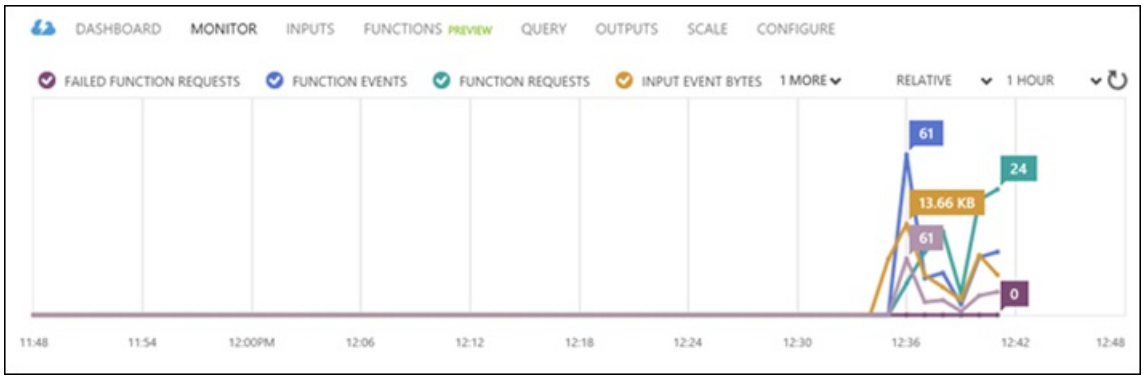
BATCH SIZE (ML LATENCY)	500 (200MS)	1,000 (200MS)	5,000 (250MS)	10,000 (300MS)	25,000 (500MS)
1 SU	2,500	5,000	20,000	30,000	50,000
3 SUs	2,500	5,000	20,000	30,000	50,000
6 SUs	2,500	5,000	20,000	30,000	50,000
12 SUs	5,000	10,000	40,000	60,000	100,000
18 SUs	7,500	15,000	60,000	90,000	150,000
24 SUs	10,000	20,000	80,000	120,000	200,000
...	...	...	...	...	...
60 SUs	25,000	50,000	200,000	300,000	500,000

By now, you should already have a good understanding of how Machine Learning functions in Stream Analytics work. You likely also understand that Stream Analytics jobs "pull" data from data sources and each "pull" returns a batch of events for the Stream Analytics job to process. How does this pull model impact the Machine Learning web service requests?

Normally, the batch size we set for Machine Learning functions won't exactly be divisible by the number of events returned by each Stream Analytics job "pull". When this occurs the Machine Learning web service will be called with "partial" batches. This is done to not incur additional job latency overhead in coalescing events from pull to pull.

New function-related monitoring metrics

In the Monitor area of a Stream Analytics job, three additional function-related metrics have been added. They are FUNCTION REQUESTS, FUNCTION EVENTS and FAILED FUNCTION REQUESTS, as shown in the graphic below.



The are defined as follows:

FUNCTION REQUESTS: The number of function requests.

FUNCTION EVENTS: The number events in the function requests.

FAILED FUNCTION REQUESTS: The number of failed function requests.

Key Takeaways

To summarize the main points, in order to scale an Stream Analytics job with Machine Learning functions, the following items must be considered:

1. The input event rate
2. The tolerated latency for the running Stream Analytics job (and thus the batch size of the Machine Learning web service requests)
3. The provisioned Stream Analytics SUs and the number of Machine Learning web service requests (the additional function-related costs)

A fully partitioned Stream Analytics query was used as an example. If a more complex query is needed the Azure Stream Analytics forum is a great resource for getting additional help from the Stream Analytics team.

Next steps

To learn more about Stream Analytics, see:

- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Rotate login credentials for inputs and outputs in Stream Analytics Jobs

1/25/2017 • 4 min to read • Edit on GitHub

Abstract

Azure Stream Analytics today doesn't allow replacing the credentials on an input/output while the job is running.

While Azure Stream Analytics does support resuming a job from last output, we wanted to share the entire process for minimizing the lag between the stopping and starting of the job and rotating the login credentials.

Part 1 - Prepare the new set of credentials:

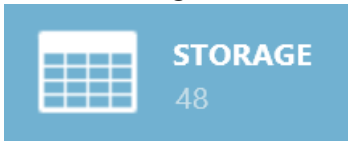
This part is applicable to the following inputs/outputs:

- Blob Storage
- Event Hubs
- SQL Database
- Table Storage

For other inputs/outputs, proceed with Part 2.

Blob storage/Table storage

1. Go to the Storage extention on the Azure Management portal:



2. Locate the storage used by your job and go into it:

3. Click the Manage Access Keys command:

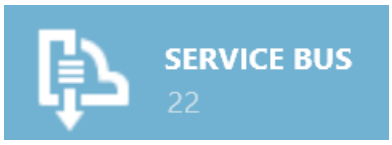
4. Between the Primary Access Key and the Secondary Access Key, **pick the one not used by your job**.
5. Hit regenerate:

6. Copy the newly generated key:

7. Continue to Part 2.

Event hubs

1. Go to the Service Bus extension on the Azure Management portal:



2. Locate the Service Bus Namespace used by your job and go into it:

3. If your job uses a shared access policy on the Service Bus Namespace, jump to step 6
4. Go to the Event Hubs tab:

5. Locate the Event Hub used by your job and go into it:

6. Go to the Configure Tab:

7. On the Policy Name drop-down, locate the shared access policy used by your job:

8. Between the Primary Key and the Secondary Key, **pick the one not used by your job**.
9. Hit regenerate:

10. Copy the newly generated key:

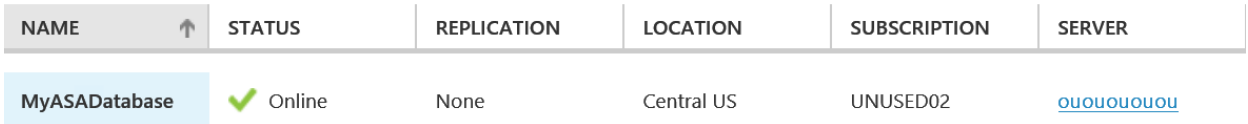
11. Continue to Part 2.

SQL Database

> **NOTE**
>
> Note: you will need to connect to the SQL Database Service. We are going to show how to do this using the management experience on the Azure Management portal but you may choose to use some client-side tool such as SQL Server Management Studio as well.

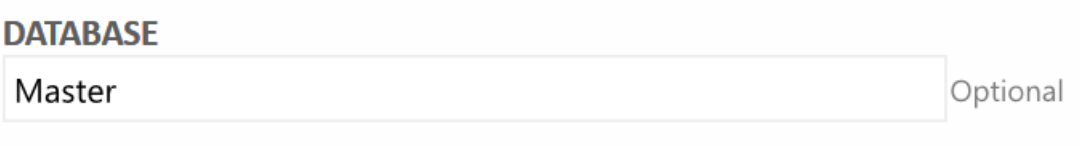
1. Go to the SQL Databases extension on the Azure Management portal:

2. Locate the SQL Database used by your job and **click on the server** link on the same line:



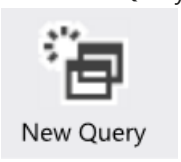
3. Click the Manage command:

4. Type Database Master:



5. Type in your User Name, Password, and click Log on:

6. Click New Query:



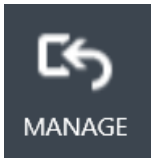
7. Type in the following query replacing with your User Name and replacing with your new password:

```
CREATE LOGIN <login_name> WITH PASSWORD = '<enterStrongPasswordHere>'
```

8. Click Run:

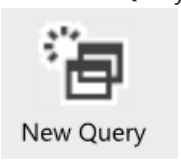
9. Go back to step 2 and this time click the database:

10. Click the Manage command:



11. type in your User Name, Password, and click Logon:

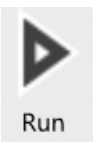
12. Click New Query:



13. Type in the following query replacing with a name by which you want to identify this login in the context of this database (you can provide the same value you gave for , for example) and replacing with your new user name:

```
CREATE USER <user_name> FROM LOGIN <login_name>
```

14. Click Run:

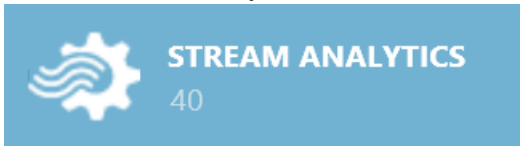


15. You should now provide your new user with the same roles and privileges your original user had.

16. Continue to Part 2.

Part 2: Stopping the Stream Analytics Job

1. Go to the Stream Analytics extension on the Azure Management portal:



2. Locate your job and go into it:

3. Go to the Inputs tab or the Outputs tab based on whether you are rotating the credentials on an Input or on an Output.

4. Click the Stop command and confirm the job has stopped:

Wait for the job to stop.

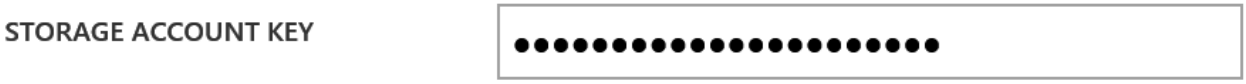
5. Locate the input/output you want to rotate credentials on and go into it:

6. Proceed to Part 3.

Part 3: Editing the credentials on the Stream Analytics Job

Blob storage/Table storage

1. Find the Storage Account Key field and paste your newly generated key into it:



2. Click the Save command and confirm saving your changes:

3. A connection test will automatically start when you save your changes, make sure that is has successfully passed.
4. Proceed to Part 4.

Event hubs

1. Find the Event Hub Policy Key field and paste your newly generated key into it:

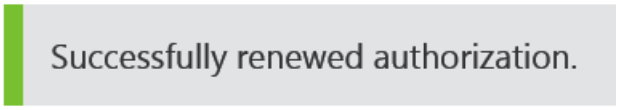
2. Click the Save command and confirm saving your changes:

3. A connection test will automatically start when you save your changes, make sure that it has successfully passed.
4. Proceed to Part 4.

Power BI

1. Click the Renew authorization:

2. You will get the following confirmation:

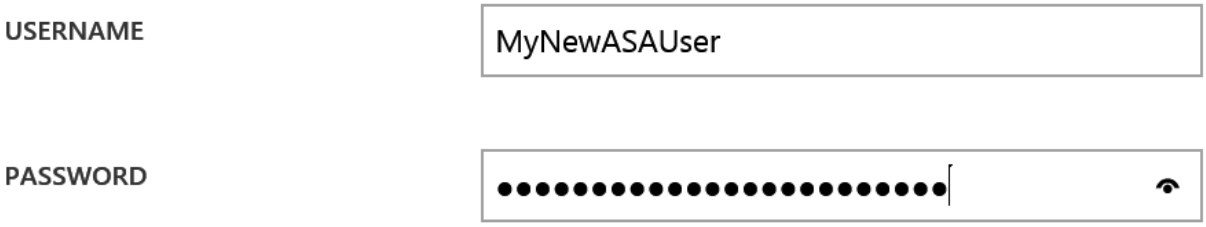


3. Click the Save command and confirm saving your changes:

4. A connection test will automatically start when you save your changes, make sure it has successfully passed.
5. Proceed to Part 4.

SQL Database

1. Find the User Name and Password fields and paste your newly created set of credentials into them:



2. Click the Save command and confirm saving your changes:

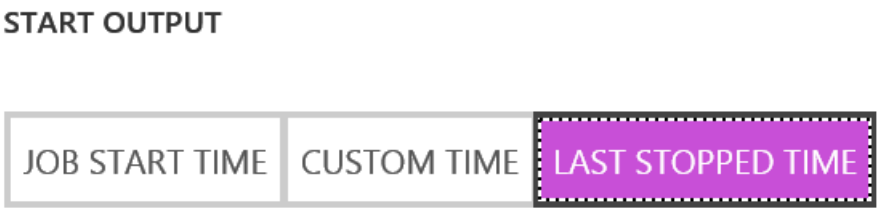
3. A connection test will automatically start when you save your changes, make sure that it has successfully passed.
4. Proceed to Part 4.

Part 4: Starting your job from last stopped time

1. Navigate away from the Input/Output:

2. Click the Start command:

3. Pick the Last Stopped Time and click OK:



4. Proceed to Part 5.

Part 5: Removing the old set of credentials

This part is applicable to the following inputs/outputs:

- Blob Storage
- Event Hubs
- SQL Database
- Table Storage

Blob storage/Table storage

Repeat Part 1 for the Access Key that was previously used by your job to renew the now unused Access Key.

Event hubs

Repeat Part 1 for the Key that was previously used by your job to renew the now unused Key.

SQL Database

1. Go back to the query window from Part 1 Step 7 and type in the following query, replacing with the User Name that was previously used by your job:
   `DROP LOGIN <previous_login_name>`
2. Click Run:

You should get the following confirmation:

```
Command(s) completed successfully.
```

Get help

For further assistance, try our Azure Stream Analytics forum

Next steps

- Introduction to Azure Stream Analytics
- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference

Stream Analytics release notes

2/1/2017 • 6 min to read • Edit on GitHub

Notes for 02/01/2017 release of Stream Analytics

This release contains the following update.

TITLE	DESCRIPTION
Introducing JavaScript User-Defined Functions (UDF)	Java User-Defined Functions are now available for additional flexibility in creating queries.
Introducing tools for Visual Studio and Stream Analytics	Tools for Visual Studio are now available for debugging and greater utility.
Introducing Diagnostic Logging	Diagnostic logging is now available for additional troubleshooting options.
Introducing GeoSpatial Functions	GeoSpatial Functions are now generally available.

Notes for 04/15/2016 release of Stream Analytics

This release contains the following update.

TITLE	DESCRIPTION
General Availability for Power BI outputs	Power BI outputs are now Generally Available. The 90 day authorization expiration for Power BI has been removed. For more information on scenarios where authorization needs to be renewed see the Renew authorization section of Creating a Power BI dashboard.

Notes for 03/03/2016 release of Stream Analytics

This release contains the following update.

TITLE	DESCRIPTION
New Stream Analytics Query Language items	SAQL now has GetType, TRY_CAST and REGEXMATCH.

Notes for 12/10/2015 release of Stream Analytics

This release contains the following update.

TITLE	DESCRIPTION
REST API version update	The REST API version has been updated to 2015-10-01. Details can be found on MSDN at Stream Analytics Management REST API Reference and Machine Learning integration in Stream Analytics.

TITLE	DESCRIPTION
Azure Machine Learning Integration	With this release comes support for Azure Machine Learning user defined functions. See the tutorial for more information as well as the general blog announcement.

Notes for 11/12/2015 release of Stream Analytics

This release contains the following update.

TITLE	DESCRIPTION
New behavior of SELECT	SELECT in Stream Analytics has been extended to allow * as a property accessor of a nested record. For further information, consult http://msdn.microsoft.com/library/mt622759.aspx.

Notes for 10/22/2015 release of Stream Analytics

This release contains the following updates.

TITLE	DESCRIPTION
Additional query language features	Stream Analytics has expanded the query language by including the following features: ABS, CEILING, EXP, FLOOR, POWER, SIGN, SQUARE, and SQRT.
Removed aggregate limitations	This release removes the limitation of 15 aggregates in a query. There is now no limit to the number of aggregates per query.
Added GROUP BY System.Timestamp feature	The GROUP BY function now allows for either window_type or System.Timestamp.
Added OFFSET for Tumbling and Hopping windows	By default, Tumbling and Hopping windows are aligned against zero time (1/1/0001 12:00:00 AM UTC). The new (optional) parameter 'offsetsize' allows specifying a custom offset (or alignment).

Notes for 09/29/2015 release of Stream Analytics

This release contains the following updates.

TITLE	DESCRIPTION
Azure IoT Suite Public Preview	Stream Analytics is included in the Public Preview of the Azure IoT Suite.
Azure Portal integration	In addition to continued presence in the Azure Management portal, Stream Analytics is now integrated in the Azure Portal. Note that Stream Analytics functionality in the Preview portal is currently a subset of the functionality offered in the Azure Management portal, without support for in-browser query testing, Power BI output configuration, and browsing to or creating new input and output resources in subscriptions you have access to.

TITLE	DESCRIPTION
Support for DocumentDB output	Stream Analytics jobs can now output to DocumentDB.
Support for IoT Hub input	Stream Analytics jobs can now ingest data from IoT Hubs.
TIMESTAMP BY for heterogeneous events	When a single data stream contains multiple event types having timestamps in different fields, you can now use TIMESTAMP BY with expressions to specify different timestamp fields for each case.

Notes for 09/10/2015 release of Stream Analytics

This release contains the following updates.

TITLE	DESCRIPTION
Support for PowerBI groups	To enable sharing data with other Power BI users, Stream Analytics jobs can now write to PowerBI groups inside your Power BI account.

Notes for 08/20/2015 release of Stream Analytics

This release contains the following updates.

TITLE	DESCRIPTION
Added LAST function	The LAST function is now available in Stream Analytics jobs, enabling you to retrieve the most recent event in an event stream within a given timeframe.
New Array functions	Array functions GetArrayElement, GetArrayElements and GetArrayLength are now available.
New Record functions	Record functions GetRecordProperties and GetRecordPropertyValue are now available.

Notes for 07/30/2015 release of Stream Analytics

This release contains the following updates.

TITLE	DESCRIPTION
Power BI Org Id decoupled from Azure Id	This feature enables Power BI output for ASA jobs under any Azure account type (Live Id or Org Id). Additionally, you can have one Org Id for your Azure account and use a different one for authorizing Power BI output.
Support for Service Bus Queues output	Service Bus Queues outputs are now available in Stream Analytics jobs.
Support for Service Bus Topics output	Service Bus Topics outputs are now available in Stream Analytics jobs.

Notes for 07/09/2015 release of Stream Analytics

This release contains the following updates.

TITLE	DESCRIPTION
Custom Blob Output Partitioning	Blob storage outputs now allow an option to specify the frequency that output blobs are written and the structure and format of the output data path folder structure.

Notes for 05/03/2015 release of Stream Analytics

This release contains the following updates.

TITLE	DESCRIPTION
Increased maximum value for Out of Order Tolerance Window	The maximum size for the Out of Order Tolerance Window is now 59:59 (MM:SS)
JSON Output Format: Line Separated or Array	Now there is an option when outputting to Blob Storage or Event Hub to output as either an array of JSON objects or by separating JSON objects with a new line.

Notes for 04/16/2015 release of Stream Analytics

TITLE	DESCRIPTION
Delay in Azure Storage account configuration	When creating a Stream Analytics job in a region for the first time, you will be prompted to create a new Storage account or specify an existing account for monitoring Stream Analytics jobs in that region. Due to latency in configuring monitoring, creating another Stream Analytics job in the same region within 30 minutes will prompt for the specifying of a second Storage account instead of showing the recently configured one in the Monitoring Storage Account drop-down. To avoid creating an unnecessary Storage account, wait 30 minutes after creating a job in a region for the first time before provisioning additional jobs in that region.
Job Upgrade	At this time, Stream Analytics does not support live edits to the definition or configuration of a running job. In order to change the input, output, query, scale or configuration of a running job, you must first stop the job.
Data types inferred from input source	If a CREATE TABLE statement is not used, the input type is derived from input format, for example all fields from CSV are string. Fields need to be converted explicitly to the right type using the CAST function in order to avoid type mismatch errors.
Missing fields are outputted as null values	Referencing a field that is not present in the input source will result in null values in the output event.
WITH statements must precede SELECT statements	In your query, SELECT statements must follow subqueries defined in WITH statements.

TITLE	DESCRIPTION
Out-of-memory issue	Streaming Analytics jobs with a large tolerance for out-of-order events and/or complex queries maintaining a large amount of state may cause the job to run out of memory, resulting in a job restart. The start and stop operations will be visible in the job's operation logs. To avoid this behavior, scale the query out across multiple partitions. In a future release, this limitation will be addressed by degrading performance on impacted jobs instead of restarting them.
Large blob inputs without payload timestamp may cause Out-of-memory issue	Consuming large files from Blob storage may cause Stream Analytics jobs to crash if a timestamp field is not specified via TIMESTAMP BY. To avoid this issue, keep each blob under 10MB in size.
SQL Database event volume limitation	When using SQL Database as an output target, very high volumes of output data may cause the Stream Analytics job to time out. To resolve this issue, either reduce the output volume by using aggregates or filter operators, or choose Azure Blob storage or Event Hubs as an output target instead.
PowerBI datasets can only contain one table	PowerBI does not support more than one table in a given dataset.

Get help

For further assistance, try our Azure Stream Analytics forum

Next steps

- Introduction to Azure Stream Analytics
- Get started using Azure Stream Analytics
- Scale Azure Stream Analytics jobs
- Azure Stream Analytics Query Language Reference
- Azure Stream Analytics Management REST API Reference