**Module Introduction – A deeper look at Stream Analytics**

**Module Introduction – A deeper look at Stream Analytics**

In this module, you will learn about

* Some of the advanced features of the Azure Stream Analytics queries
* Analytics topologies and message routing concepts
* Device management and how it relates to analytics

This module introduces you to the benefits of using stream analytics and it introduces advanced query language topics. You will set up an event hub as an alternate method of building an IoT architecture, and you will be introduced to some of the device management capabilities of the Azure IoT Hub.

During this module, you will complete the following hands-on labs:

* Lab 1: Constructing Analytics Queries
* Lab 2: Managing Analytics Topologies
* Lab 3: Device Management And Analytics

**Module 4 setup**

This module utilizes leverages several of the applications and technologies that you created for previous modules.

If you have deleted any of them for cost considerations, you will need to recreate them, for use in this module. The labs assume the applications are available.

Requirements:

* Azure IoT Hub
* Azure Solution accelerator (device simulation)
  + Some of the IoT Hub routing labs require real-time processing of stream data
* Azure Data Lake Storage
* Azure Data Lake Analytics
* Visual Studio Code

**Create Blob storage for stream analytics reference data**

Stream Analytics uses Azure Blob storage as the storage layer for Reference Data. Your first step in this lesson will be to create a storage account and a blob container. The supported formats are CSV and JSON; you can create either type of file and upload it to a blob container. In this lesson, we will use a CSV file with metadata about our virtual wind turbines.

**IMPORTANT:** You will use a comma separated value file (IoTTurbineStaticData.csv) for your reference data. In order for it to work with your streaming data, you will need to change the value of the first column (ConnectionDeviceId), to match the ConnectionDeviceId json value in your simulated data records. The value from the device simulation is randomized, each time you start a new simulation. A typical value looks like this: 122f2c65-f8b9-4bb8-9314-fbc9fd1cf10e.9471b892-57c8-4260-9172-45533a79cc6f.9. So essentially, the format is: <GUID>.<GUID>.<DEVICENUMBER>.

You will need to find out the value of those first two guids from your device simulation. You can find them on the device simulation landing page, or you can extract the values from the IoT Hub by sampling the data from your Azure Stream Analytics job. (See Module 1, Lesson 1, Steps 3 and 4). If you do not correctly match the two ConnectionDeviceId values, the rest of the module will not work. So let's start there

1. Download this file to your local machine:

[IoTTurbineStaticData.csv](https://prod-edxapp.edx-cdn.org/assets/courseware/v1/e32ab2698f62e42ddc09f6550fe14b9e/asset-v1:Microsoft+DEV326x+1T2019+type@asset+block/IoTTurbineStaticData.csv)

1. Open the file with you text editor

Inside, you will find rows that look like this:

ConnectionDeviceId,Manufacturer,Height,WindFarm,Latitude,Longitude,TurbineName

<REPLACE\_WITH\_GUID1>.<REPLACE\_WITH\_GUID2>.1,Bunison,80,Northern California,40.678,-122.3,NorCal.Bunison.80.0.3

<REPLACE\_WITH\_GUID1>.<REPLACE\_WITH\_GUID2>.2,Veztas,160,Northern California,40.678,-122.3,NorCal.Veztas.160.0.82

...

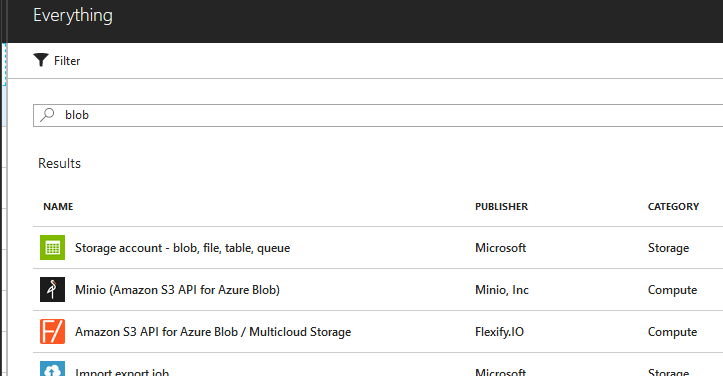
1. Find the format of the ConnectionDeviceId in your simulated data

The value will probably look something like this: 122f2c65-f8b9-4bb8-9314-fbc9fd1cf10e.9471b892-57c8-4260-9172-45533a79cc6f.9

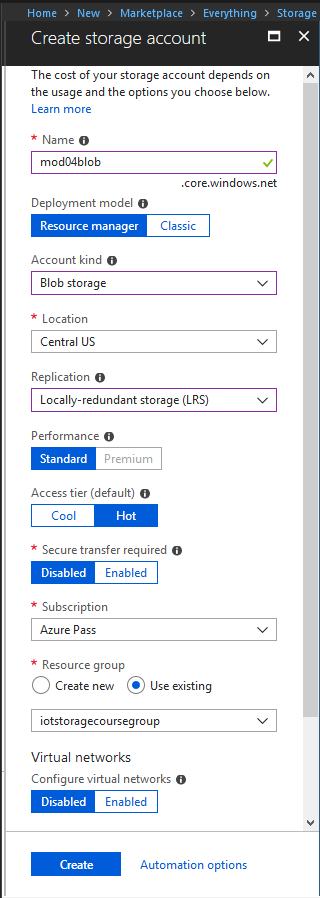
1. Replace all instances of the placeholder <REPLACE\_WITH\_GUID1> with the first guid
2. Replace all instances of the placeholder <REPLACE\_WITH\_GUID2> with the second guid
3. Save the file

Now you are ready to set up your reference data in blob storage.

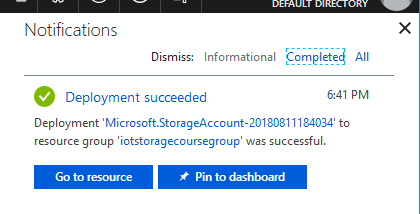
1. In the Azure Portal, click **Create a resource** in the upper-left corner.
2. In the **Search the Marketplace** box, type **blob**
3. Select **Storage account – blob, file, table, queue** from the results.
4. Click **Create** on the next screen.



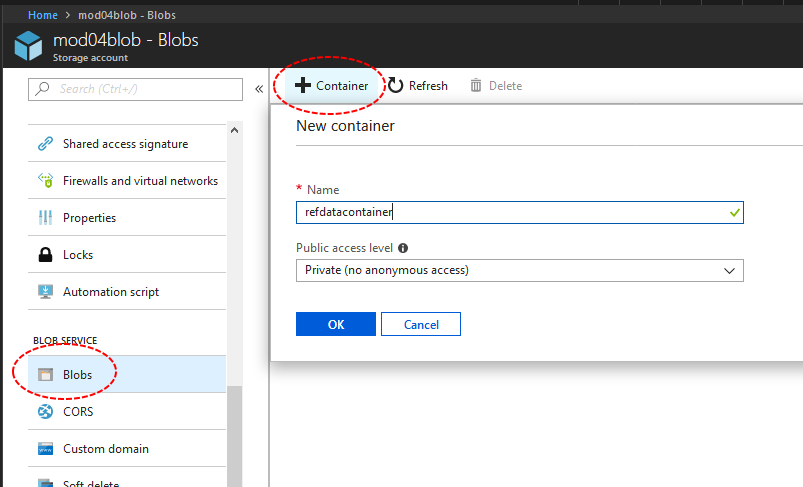
1. On the **Create storage account** screen, under **Name**, enter a unique name such as **mod04blobXX**.
2. Under **Account kind**, select **Blob storage**
3. Under **Location**, select the region you have been using for other resources.
4. Under **Replication**, select **Locally-redundant storage (LRS)**
5. Under **Access tier**, click **Hot**
6. Under **Resource group**, click **Use existing** and select the group you’ve been using for the resources.
7. Click **Create**.



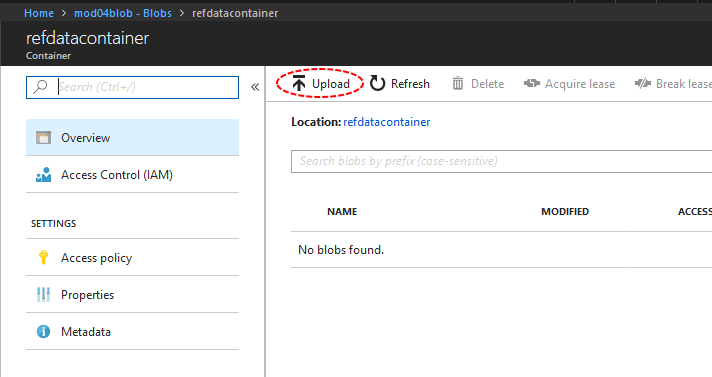
1. Watch your notifications and click **Go to resource** when the storage account has been deployed.



1. On the **Storage Account** page, select the **Blobs** blade.
2. Click **Container** to create a new container.
3. Under **Name** type **refdatacontainer**, and then click **OK**

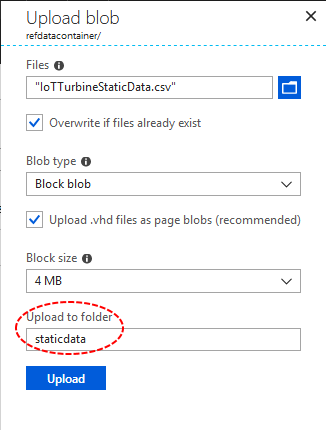


1. On the **Container** page, click **Upload** to upload your turbine metadata csv file.

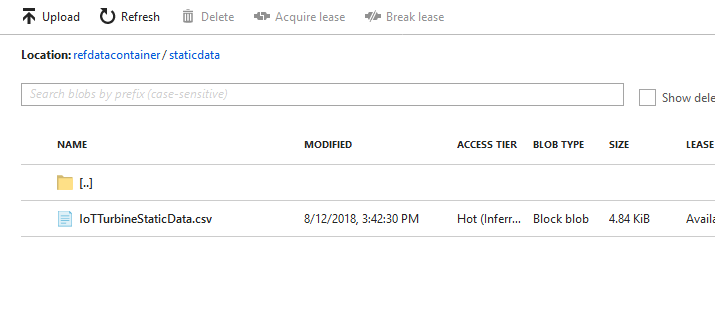


1. On the **Upload blob** page, click **Advanced** to expand the page.
2. Under **Files**, click the folder icon and locate the **IoTTurbineStaticData.csv** file that is included with this module and select it.
3. Under the **Upload to folder** box, enter **staticdata**

This will create a folder within your blob and upload the file there.



The landing page will now indicate that your **IoTTurbineStaticData.csv** file is located in **refdatacontainer / staticdata**



**Summary**

In this lesson, you created blob storage that you will use for reference data in an Azure Stream Analytics job. You uploaded a csv file with reference data into a folder within blob storage.

## Query Time Windows

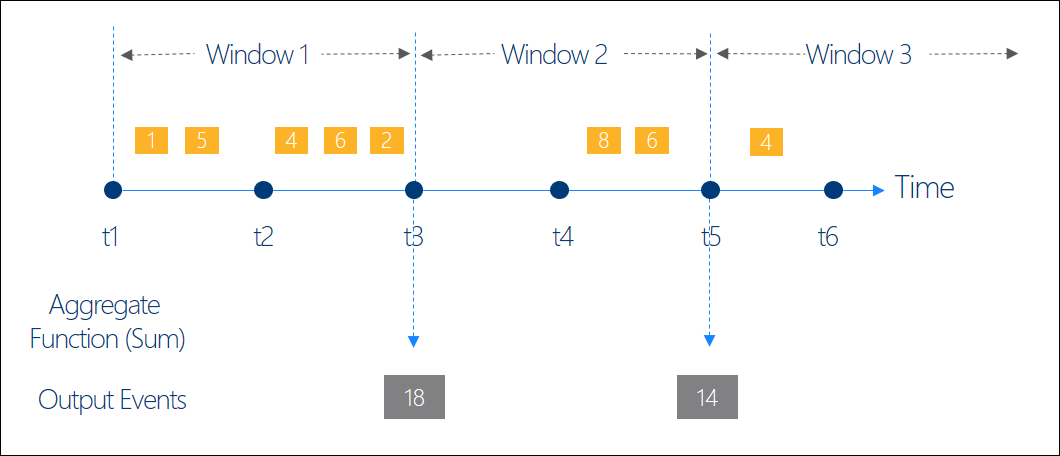
### Query telemetry data with time window functions

1. Streaming overview

Streaming data, by definition, has a time element. IoT telemetry data nearly always has a timestamp for when it is created. The same data gets an additional timestamp when it is processed. Processing stream data consistently is one of the bigger challenges of working with time series data. Stream Analytics has native support for windowing functions, enabling developers to author complex stream processing jobs with familiar SQL syntax. This is a differentiator from other analytics packages where consolidating time series data can often be difficult.

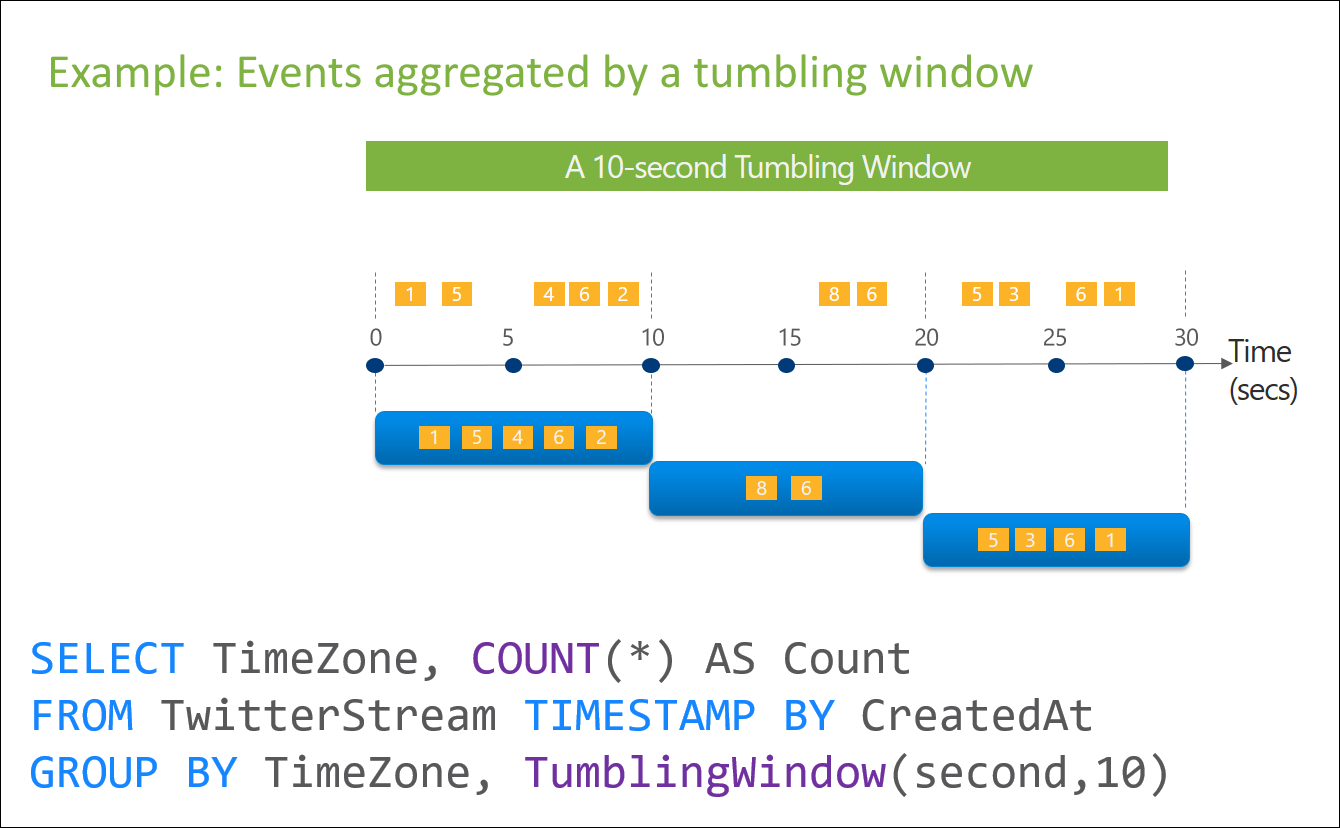
There are four kinds of temporal windows to choose from with Azure Stream Analytics: Tumbling, Hopping, Sliding, and Session windows. You use the window functions in the GROUP BY clause of the query syntax, in your Stream Analytics jobs.

All the windowing operations output results at the end of the window. The output of the window is a single event, based on the aggregate function used. The output event has the time stamp of the end of the window and all window functions are defined with a fixed length.

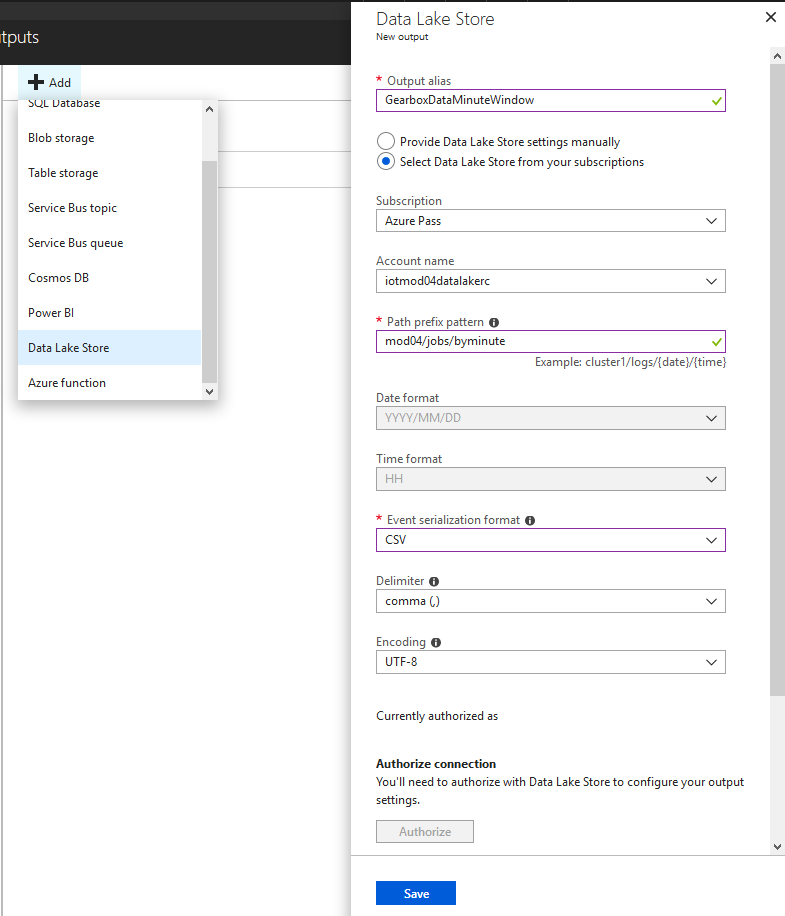


In this step, you will write some queries **TumblingWindow**; the most straightforward type of time window to understand. With a tumbling window, there is a predefined length of time (anything from microseconds to days). If an event occurs within that window, it is aggregated with the other events that happen within that window. There is no overlap, and an event can only belong to one window.

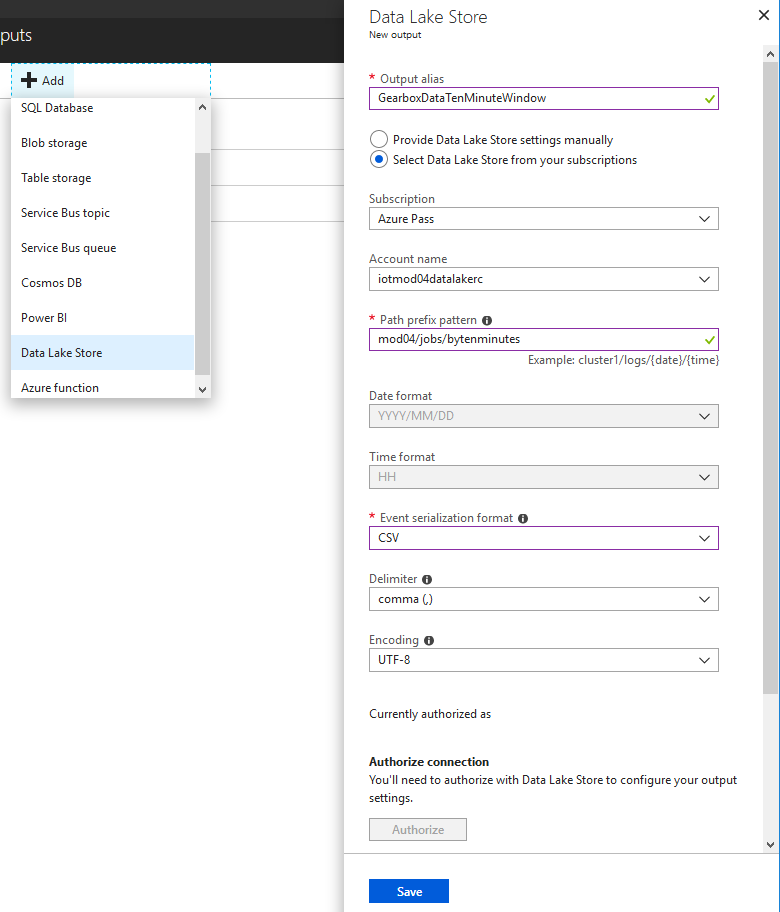
There are many use cases for this kind of query. One of the main ones is sheer volume of data. If a temperature sensor emits data every few seconds, but temperature variations only matter over the course of minutes, it makes sense to aggregate the data with windowing functions.



1. Navigate to the **Outputs** window of the Azure Stream Analytics job you created in the previous lessons.
2. Click **Add**
3. Select **Data Lake Store**
4. In the **New Output** window, under **Output alias**, type **GearboxDataMinuteWindow**
5. Click the **Select Data Lake Store from your subscriptions** radio option.
6. Under **Account name**, select the Data Lake from your account.
7. Under **Path prefix pattern**, type **mod04/jobs/byminute**
8. Under **Event serialization format**, select **CSV**
9. Click **Authorize** to allow access to the Data Lake store.
10. Click **Save**



1. On the **Outputs** window, click **Add**
2. Select **Data Lake Store** again.
3. In the **New Output** window, under **Output alias**, type **GearboxDataTenMinuteWindow**
4. Click the **Select Data Lake Store from your subscriptions** radio option.
5. Under **Account name**, select the Data Lake from your account.
6. Under **Path prefix pattern**, type **mod04/jobs/bytenminutes**
7. Under **Event serialization format**, select **CSV**
8. Click **Authorize** to allow access to the Data Lake store.
9. Click **Save**

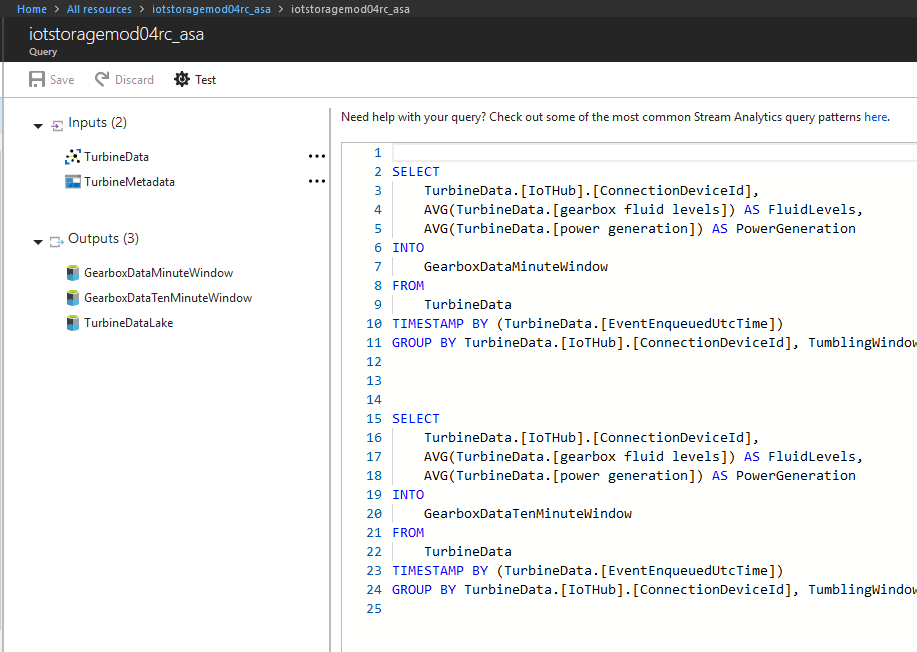


For the Contoso Wind Power scenario, we have a vendor who is in charge of maintaining the gear boxes on our wind turbines. The vendor is not concerned with the fluid levels every ten seconds (like our telemetry data sends). In fact, if we send that much data, the vendor's system will get overwhelmed. So you will generate some data aggregated over a longer time window.

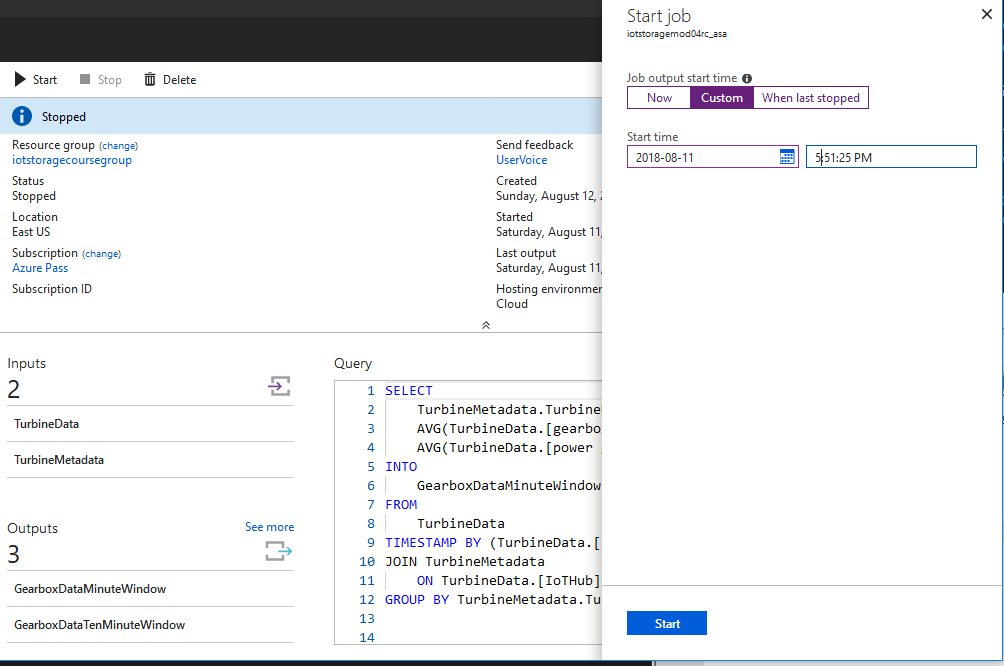
1. On the **Query Page**, replace the query text with the following:
2. SELECT
3. TurbineData.[IoTHub].[ConnectionDeviceId],
4. AVG(TurbineData.[gearbox fluid levels]) AS FluidLevels,
5. AVG(TurbineData.[power generation]) AS PowerGeneration
6. INTO
7. GearboxDataMinuteWindow
8. FROM
9. TurbineData
10. TIMESTAMP BY (TurbineData.[EventEnqueuedUtcTime])
11. GROUP BY TurbineData.[IoTHub].[ConnectionDeviceId], TumblingWindow(second, 60)
12. SELECT
13. TurbineData.[IoTHub].[ConnectionDeviceId],
14. AVG(TurbineData.[gearbox fluid levels]) AS FluidLevels,
15. AVG(TurbineData.[power generation]) AS PowerGeneration
16. INTO
17. GearboxDataTenMinuteWindow
18. FROM
19. TurbineData
20. TIMESTAMP BY (TurbineData.[EventEnqueuedUtcTime])
21. GROUP BY TurbineData.[IoTHub].[ConnectionDeviceId], TumblingWindow(minute, 10)

Notice that the queries aggregate average readings for fluid levels and power generation. The queries are identical, except for the tumbling window. The first is one minute TumblingWindow(second, 60) and the second is 10 minutes TumblingWindow(minute, 10).

1. Click **Save**

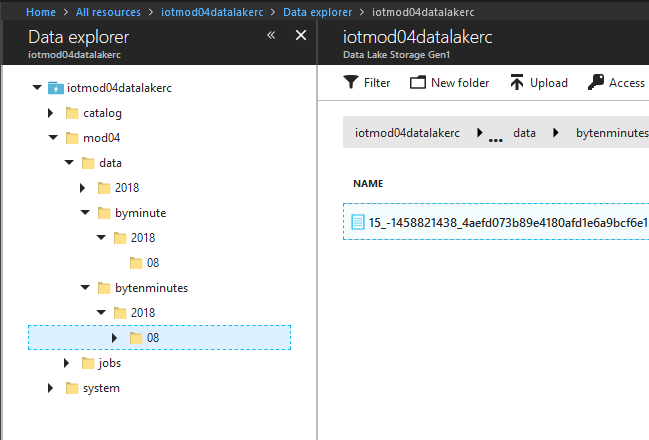


1. On the stream analytics page, click **Start**, to start the job.
2. On the **Start job** page, under **Job output start time**, click **Custom**
3. Under **Start time**, enter a date and time when you started streaming data from your IoT hub into your stream analytics job.



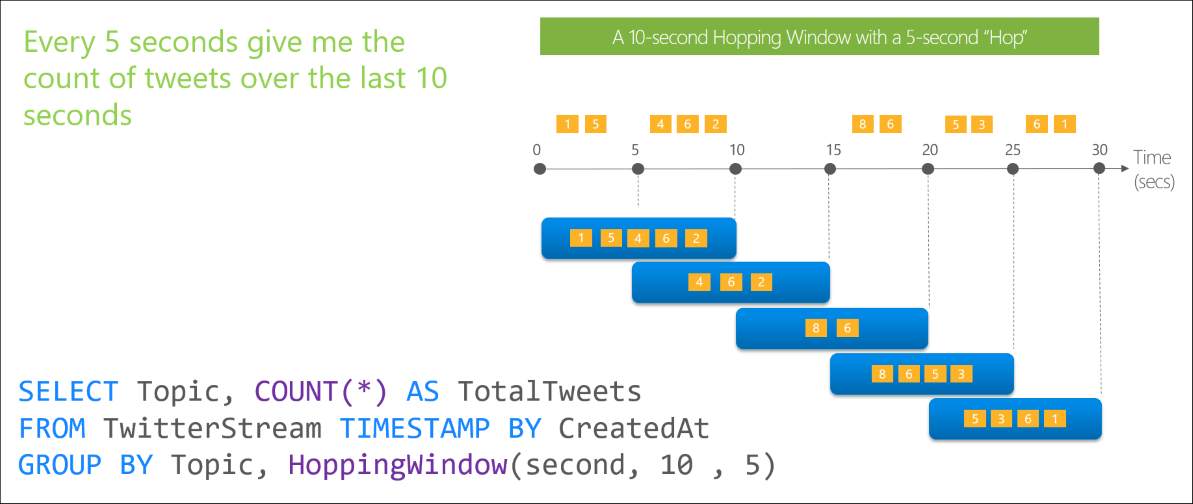
1. Once the job completes, navigate to the **Data explorer** page of your Azure Data Lake account.
2. You should see two new folder paths **byminute** and **bytenminutes**
3. Download the csv files in these folder paths and compare them.

The file with the 10 minute window will have about 10x fewer rows than the 1 minute file. The time window is correspondingly longer.

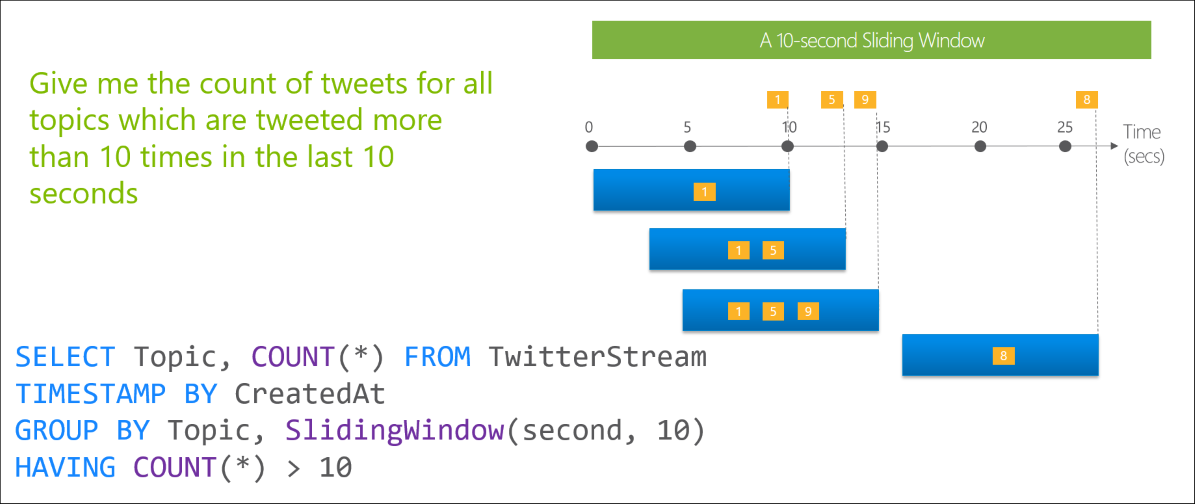


The simulated data we have does not lend itself to the other types of time windows, but here are some descriptions and examples:

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, specify the hop size to be the same as the window size.



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.



Deciding which type of window to use is highly dependent on your use case.

### Summary

In this lesson, you learned about the time element of streaming data and some of the tools Azure Stream Analytics has for working with that data. You wrote queries with a tumbling window, to compare the results of using different time windows.