# Course Overview

## Course Overview

Hi everyone. My name is Reza Salehi, and welcome to my course, Building Streaming Data Pipelines in Microsoft Azure. I am a cloud consultant and trainer. There are many sources providing live data streams from the stock market and banking applications to industrial sensors and connected IoT devices. This live data needs to get processed and \_\_\_\_\_ active often in real time. In this course, we are going to use Azure Stream Analytics to process live data streams.

Some of the major topics that we will cover include

* understanding windowing functions,
* configuring the stream and reference inputs,
* the Stream Analytics query language, and finally,
* integrating the output with Power BI.

By the end of this course, you will know Azure Stream Analytics well enough to build your own live data processing pipeline. Before beginning the course, you should be familiar with Azure Portal and T-SQL. I hope you'll join me on this journey to learn Azure Stream Analytics with the Building Streaming Data Pipelines in Microsoft Azure course, at Pluralsight.

# Azure Stream Analytics Overview

## Overview

Hi everyone. I am Reza Salehi. In this course, we are going to learn how to Build Streaming Data Pipelines in Microsoft Azure, and as you can guess from the title of this module, we are going to use Azure Stream Analytics service to do so. It is important to have a solid understanding of live data processing and its use cases. We are going to start this module by explaining what live data processing is, and we will mention a few use cases.

The specific systems need to be built to process live data, and building these systems is challenging. We are going to go over a few challenges you will face when building live data processing systems. We will also see which technology choices you have if you like to build your live data processing system in Microsoft Azure, and this brings us to the main topic of this course. Throughout this course, we are going to use Azure Stream Analytics to process live data. I will introduce this service and its purpose. It is also important to understand which kind of inputs and outputs this service supports. And before moving onto the demo, there is one more important concept we need to cover, and it is the concept of windowing. In any live data processing system, you need to have an easy way to find live events within a specific time window, and this is exactly what Azure Stream Analytics windowing does. We will go over four different windowing options you have when processing live data. And finally, in the module's demo, we are going to provision an Azure Stream Analytics instance. We will create a stream analytics job in this instance, and we will use it to process some data. This is a very interesting course, so let's get started.

## What Is Live Event Processing?

So before turning our focus to Azure Stream Analytics, let's have a closer look into live data processing concept. There are a few common components in any live data processing. You have some sort of live data stream coming into your system, and we will get back into examples shortly. Your system is going to ingest these live data streams and process them in real time. The end result or the output of this processing will be sent to the output of the system, and you can act on this output. For example, imagine you have a few heat sensors in a nuclear power plant. These heat sensors constantly monitor the temperature of your reactor core and send it to your system. Any type of connected device can also generate live data. These fall into the category of Internet of Things. The stock market is also generating huge amounts of live data streams, which can be ingested. Another example of live data is the audit information created for an online banking application. Users can log into this online banking platform and perform financial tasks. Every financial task will be logged and can be sent to any system processing this data in real time. Logs created in real time are an important source to many live data processing systems. And in general, any event created in real time can be an input to live data processing system, so you have figured out your live data stream, so your live data processing system is going to ingest this live data and process the data in real time. And what do we mean by processing? What kind of processing you might want to do on a live data stream. There are many scenarios, and really, the sky is the limit here. In case of heat sensors from a nuclear power plant, you might process that live data and look for sudden changes in the reactor core temperature, and you might want to act on it. A live data processing system can help you achieve that. For the online banking example, you might look for anomalies in the online banking audit logs. For example, if a user is suddenly logging in from a different country or a huge amount of money is being transferred, in general, you can use live data processing systems to look for any pattern that you choose in the live data stream coming in. These systems can also be used to aggregate the data. In the nuclear power plant case, you might write a query that ingests the live data about the temperature and gives you an average of the core temperature in 5-minute windows. And as I said, the sky is the limit here. So far, you have ingested live data streams and processed them. What is the next step? All live data processing systems support some sort of output, and these outputs can be used to link these systems into target systems. For example, in the case of sudden changes in the core reactor temperature, you might want to alert the engineering team. In case of anomalies, in an online banking session, you might want to abort the session before it is too late. In case of sudden changes in the price of a specific stock, you might want to send emails or other sort of alerts to specific people. And finally, the output of your live data processing can be stored in a database for later analysis and processing. So just to recap the use cases, using a live data processing system, you can generate alerts when the power plant core temperature suddenly changes. This is very important to prevent any catastrophe. You can block malicious online banking users before it is too late. You can abort their session or block that specific IP address. Or in the stock market scenario, you can build a system that automatically buys or sells a stock in real time based on the patterns you specify. And finally, you can develop a system that processes security logs and prevents data security breaches in real time. For example, Azure SQL database has an advanced threat detection feature. This feature is constantly monitoring as your SQL database logs and acts accordingly if any anomalies or security problem is spotted. Now that we understand the use cases for live data processing, let's take a look at the challenges we are facing when developing these systems.

Live Data Processing Challenges

Due to the nature of live data processing systems, the data ingestion, processing, and output should happen in real‑time. This introduces a few challenges for us. First, we need to make sure your live data processing system has support for high volumes of data input. In some scenarios, your system needs to ingest megabytes, hundreds of megabytes, or even gigabytes of data input per second, and ingesting the inputs needs some processing power. The data needs to be ingested and made ready to the processing engine, so you need to make sure your system has enough processing power so data ingestion is not interrupted. This means, in most cases, live data processing systems cannot be deployed to commodity machines. The machines should be powerful and scalable. Your live data processing system produces outputs. You need to make sure your output storage has enough right bandwidth so data can be outputted without any issues or interruptions. And last, but not least, you need to act quickly on the output of your processing. If there is a sudden rise of your reactor's core temperature, you cannot wait for hours or even minutes to act on it. You need to immediately issue alerts to the engineering team so they can address the issue. Same goals with most other scenarios, including stock markets, signals from IoT devices, and online banking. If your live data processing system detects anomalies in an online banking session, you cannot wait for hours or minutes. You need to act on it as quickly as possible, maybe in the matter of seconds so any damage is prevented. So, as you can see, we are facing significant challenges when implementing live data processing. And here is when Microsoft Azure comes to the rescue. Let's take a look at a few options Microsoft Azure gives you to implement live data processing. You can use HDInsight with Spark Streaming for your live data processing scenarios. HDInsight with Storm is another option, as well as Apache Spark in Azure Databricks. Some other services such as Azure Functions can be used to process live data. Azure Functions can be triggered by Service Bus queues or Event Grid. So, for example, you can ingest live data from Event Grid and invoke an instance of Azure Function per each data item. Azure WebJobs is another technology, which can be used to process live data. And finally, Azure Stream Analytics is a service specifically designed for live data processing, and it is the main focus for the rest of this module and course, so let's have a closer look into Azure Stream Analytics.

Introducing Azure Stream Analytics

And finally, the time has come to focus our attention to Azure Stream Analytics. This service will be the main focus of this course, so what is Azure Stream Analytics? As the name suggests, this service is offered by Microsoft Azure, and it is a fully-managed, real-time analytics service designed to process fast-moving streams of data. So, as you can see, it is a managed service. This means you don't need to worry about provisioning virtual machines or monitoring performance. It is a real-time analytics service. This means it can ingest the live data, process it on the fly, and generate outputs in near real time, and finally, it can process fast-moving streams of data. Data ingestion, volume, and processing is not a problem. Now let's see how the data stream flows into Azure Stream Analytics. First, you need to provision an Azure Stream Analytics job. Azure Stream Analytics job can accept streaming data from three different input types. The first one is Event Hubs, and the application can be configured to send events to Azure Event Hubs. For example, you might have an application which monitors the stock market prices. This application can be configured to send the stock prices in specific intervals, for example, every 1 second. These events can be sent to Event Hubs and in turn ingested into Azure Stream Analytics. The other example will be our power plant. The heat sensors can be configured to frequently send temperature events to Azure Event Hubs. These events also will be ingested into Azure Stream Analytics and can be processed later. Azure IoT Hub is another streaming input type which can be used with Azure Stream Analytics. Azure IoT Hub is an event ingestion service, which is highly optimized towards IoT scenarios. You can have drones or robots frequently send events to IoT Hub, which in turn will be ingested into Azure Stream Analytics. And finally, Azure Blob store can be used as a source of streaming data into Azure Stream Analytics, and this option is great for lock files. You can configure your systems to create audit and system usage files in form of blob storage, which in turn can be ingested to Azure Stream Analytics. So, as you can see, we have three types of streaming data, which can be used as input. There is also another type of input, which is called reference data. For the reference data, you can use Azure Blob storage and Azure SQL Database. We are going to take a look at a streaming data input, a reference data input in the module's demo. So you have your data figured out. This data will be ingested by Azure Stream Analytics. The service is going to process the data using a query you provide. Azure machine learning can also be used in the process if you need to involve some sort of machine-learning scenario when processing your data. So after your output is ready, it will be sent to any output, which you configure for your Azure Stream Analytics service, and there are a few of them. You can configure Azure Stream Analytics to write outputs into Azure blobs, Data Lake Gen 1, and Data Lake storage Gen 2. Azure SQL Database and Azure SQL Data Warehouse can also be used as the output for Azure Stream Analytics. Azure Event Hubs is another output, which can be used for Azure Stream Analytics, and this is extremely useful because you can have other services subscribing to Event Hubs. So each time an output from Azure Stream Analytics is ready, the target system can go ahead, take the output, and act on it. And finally, Azure Stream Analytics can be configured to send outputs to Power BI. Power BI is a product created by Microsoft and can be used to visualize your data. Using this service, you can visualize the output of Azure Stream Analytics and perform more processing on it if necessary. So now let's see how we can provision our own instance of Azure Stream Analytics. First of all, you need to figure out your input. As you remember, one of the inputs is Event Hubs, so you can go ahead and provision a new instance of Event Hubs in the portal or programmatically. Same goes with the IoT Hub. You can go ahead and create a new instance of IoT Hub in the Azure portal or programmatically. And finally, as you remember, Azure Blob storage can be used as an input for Azure Stream Analytics. To be able to use blobs, you need to create a new storage account and then create a blob container inside it. We are going to take a look at this option in the module's demo. After you have your input provisioned, you are ready to create a new Stream Analytics job. Simply search for this resource type in the Azure portal and click on Create. We are going to take a look at provisioning of a new instance of a Stream Analytics job in the module's demo.

What Is Time Windowing?

Time is the most important variable in any live data processing system, and we need to make sure we have a solid understanding of the concept time windowing. Let's just start with a simple database example. Imagine you have a table in your database. This table has three columns, Name, Date, and TimeOff, and as you can see, each row in this table represents a single TimeOff instance taken by an employee. As you can see, John took 1.5 hours off on Monday, Mary took 3 hours on Tuesday, Reza took 2 hours on the same day, and finally, John took 3 hours on Friday again. Now let's say the manager needs to get the report at the end of the week so he or she can see how many time off hours each employee took. There is a simple SQL query that you can use to get this information back. As you can see, I'm aggregating all the time offs using the SUM function, and I'm grouping my aggregation per each person, and this is the end result I'm going to get. So as we can see, John took 4.5 hours off in the past week. Now let's think about live data. I have a heat sensor in my power plant. This heat sensor is sending me temperature information every 5 seconds. This information is ingested by a live data processing system. To be able to monitor my power plant, I need to have the average of all the temperatures in the past 10 seconds, and I need to get the information every 10 seconds. The question is, what kind of query I need to write to get this information back, and this is where the concept of time windowing comes in. We are going to define a time window of 10 seconds, and we are going to use the average function for each 10 second time window. And this is the final data I am going to get back. As we can see, for the first 10 seconds, my average is 109 degrees and change, and this is the average of the first three sensors data I got. I am going to get the second average at the 20-second mark, and the average is 117 degrees. And finally, the last average will show up at the 30-second mark with the amount of 191.97 degrees. And as long as I have data coming from the sensor, I am going to get these averages every 10 seconds. So let's take a closer look at the time windowing for Azure Stream Analytics. As you saw, each data event has a timestamp, and by default, this timestamp is equal to the time the event was ingested. There is a need in many live data processing systems to perform an operation on events falling in the same time window. In our case, the time window was 10 seconds and the operation was average. There are other operations such as count, minimum, maximum, and much more. All live data processing systems offer an easy method, which can be used to find these subsets of data events. In our case, Azure Stream Analytics achieves this through windowing functions. There are four different window types that you can use in Azure Stream Analytics. There is Tumbling window, Hopping window, Sliding window, and finally, Session window. Now let's go ahead and take a closer look into these windowing functions.

Tumbling Window and Hopping Window

The first window type you can use in Azure Stream Analytics is a Tumbling window. The size of a Tumbling window is fixed and there is no overlapping between consequent windows. Imagine I have these events coming into my Azure Stream Analytics. As you can see, I have three events happening between the 0-second and 5-second mark. I have the event number for happening between 5-second and the 10-second mark, and finally, I have two events happening between the 10-second mark and the 15- second mark. My goal is to select all the events in 5-second windows. So, in this case, I want to have three rows in my result set, the first row containing event number 1, 2, and 3, the second row containing event number 4, and the third row containing event number 5 and 6. This is the query I'm going to write in Azure Stream Analytics. As you can see, I am using a function called TumblingWindow, and I'm using it with a GROUP BY clause. This function accepts two parameters. The first parameter is a unit of time, in my case, second. I can also use microsecond, millisecond, minutes, hours, and days, and the second number is the length of the window in that unit size. So as you can see here, I'm creating tumbling windows, and each of them is 5 seconds long. So if you are familiar with T-SQL, you know that I can't use aggregate functions if I'm using GROUP BY in my query. And in my case, I'm getting the count of all the events within each Tumbling window. So now let's go and see running this query how my Tumbling windows look like. The first Tumbling window has three events in it, e1, e2, and e3. The second Tumbling window has only 1 event, and finally, my third Tumbling window has two events in it, event 5 and event 6. As you can see, the size of my windows are fixed and is equal to 5 seconds, and the windows are not overlapping. So running this query, I'll get three rows in my output with the values of 3, 1, and 2, which is the number of events happening in these time windows. And one last thing to note is that my result set is not limited to these three rows. As long as I'm ingesting events, I am going to get these numbers every 5 seconds, so I can send them to any output for later processing. The second type of window you can use in Azure Stream Analytics is a Hopping window. As you can see, I have a list of time series events coming in. I have 1 event in the first 5 seconds, 2 events within the 5-second and the 10- second mark, and so on. Similar to a Tumbling window, the size of a Hopping window is fixed, but the difference is the consequent windows are overlapping. In this scenario, my goal is to get the events within a 10-second window, but to get my next window, I'm not going to jump for 10 seconds. I am going to only jump for 5 seconds. This is the query I'm going to write. As you can see, the function we are going to use is called Hopping window, and it accepts three parameters. The first one is a unit of time, the second one is the size of the window, in my case 10 seconds, and the third one is the hopping size in seconds, in my case, 5 seconds. Now let's see how my Hopping windows look. The first window has three events in it, event 1, event 2, and event 3. To get the second window, I am only \_\_\_\_\_ for 5 seconds, but I'm looking back for 10 seconds. In this case, my second window has three events in it, events 2, 3, and 4. And as you can see, the first window and second window are overlapping because events number 2 and 3 exist in both windows, so I can proceed with the third window, fourth window, and fifth window. So running this query, I am going to get five rows back with the values 3, 3, 2, 2, and finally, 2.

Sliding Window and Session Window

Sliding window is the third type of window you can use with Azure Stream Analytics. The size of the Sliding windows are fixed, but I'm going to only get new windows if new events are happening. This is a sample input I'm ingesting into my Azure Stream Analytics. As you can see, I have five events here. In this scenario, my goal is to get a new window each time a new event happens. And for each window, I want to go 10 seconds back. As you can see, the function I'm going to use in my query is called a Sliding window, and it accepts two parameters. The first parameter is a time minute, and as you remember, I can also use microseconds, milliseconds, minutes, hours, and days. So let's see how my windows look like when running this query. The first window is always an exception, so I'm going to get a 10-second window at the 10-second mark if there is any event present before the 10- second mark. So the second window ends at the e3 mark and goes back for 10 seconds, and this is why event number 2 is also included in my window. The third window is created when the event number 4 is generated and goes back for 10 seconds. So I have event 3 and event 4 in that window. And as you can see, my second and third window are overlapping. And finally, my last window for this input is created when the fifth event is occurring and goes back for 10 seconds. And because there is more than 10 seconds difference between the time event number 4 and event number 5 occurred, my last window doesn't overlap with window before it, and it has only one event in it. So running this query, I am going to get four rows back, 2, 2, 2, and finally, 1. By the time we are recording this course, a Session window is the last type of window you can use with Azure Stream Analytics. The Session windows are not fixed size, and the consequent windows might not overlap. A Session window is only created when there are new events present, so there is no windows in the moments of silence. This is a sample input I'm ingesting into my Azure Stream Analytics. My goal is to only have new windows if there are new events present. I want the window to keep extending as long as there are events happening, but I want the window to close if there is silence for more than 5 seconds. As you can see, with this logic, each window can keep extending if there are events happening in less than 5 seconds of the previous one. To prevent the window size to extend without a limit, I am going to put a 10-second limit as the maximum size of my window, so let's see how we can achieve this. I am going to use a session window function in my Azure Stream Analytics query. As you can see, this function accepts three parameters. The first parameter is the unit of time. The second parameter is the time out, in my case, 5 seconds. So if there is silence for more than 5 seconds, I want to close the current window. And the third parameter is the maximum size I want for my window, in my case, 10 seconds. So if the size of the window exceeds 10 seconds, the current window should close. Let's see how my windows look like. The first window opens when the first event happens at the e1 mark right before 5 seconds. As soon as the first window opens, its countdown starts for 5 seconds. So if there is no new event happening in the next 5 seconds, the window will close. However, in my case, event number 2 happens before the 5-second timeout expires, so my window extends. But right after that, I have silence for more than 5 seconds, and that's why my first window closes at the 12-second mark. So when do you think the second window opens? You're right. At the e3 mark. And as we can see, these two windows don't overlap. The second window opens at the e3 mark and keeps extending because I have event number 4, event number 5, and event number 6 happening before the 5-second timeout expires. So you might ask why this window is so long, it's over 15 seconds. I set the maximum time of my window to be 10 seconds. Azure Stream Analytics only checks for the size of a Session window at multiples of the maximum size. In my case, I set the maximum size to 10 seconds, so Azure Stream Analytics is only checking for the maximum size of the window at the 10 seconds, 20 seconds, 30 seconds mark. And this is why my second window keeps extending. Logically, my window should have been closed before the 25-second mark, but there is no check for the maximum window size at that time, so the window keeps extending before the 30-second mark, and this is why the second window is over 15 seconds long and contains 5 events. So running this query, I am going to get two rows back. The first row with two events and the second row with five events. We are going to use Azure Stream windowing in our queries in the next modules.

Demo: Creating Our First Azure Stream Analytics Job

There is a lot we covered in this module. Now let's go ahead and see Azure Stream Analytics in action. We are going to provision a new Azure Stream Analytics job. After that, we are going to use this job to process a blob storage input. I am logged into the Azure portal. Our goal in this demo is to take some sample data into Azure's Stream Analytics, process it, and then create an output and write it into a new blob storage. I have created this sample data, which we are going to use as the input for Azure Stream Analytics. These are temperature information I generated using a custom application. And as you can see, each temperature record has an ID, the temperature in Celsius, sensor ID, and the time of the event. And I have about 50 of these events in this file. I also made this file available in the course files if you want to go ahead and try it on your own. First, let's go ahead and create a new storage account. We are going to use this storage account for the input and output of Azure Stream Analytics. I already book marked the storage accounts so I could click on it or I could click on Create a resource, and search for storage, and click on a storage account. Let's click on Create. I am going to create a resource group for the purpose of this demo, so let's name the resource group, click on OK, and also let's name the storage account streamstorage01. Looks like this name is taken, so I'm going to add ps at the end, and I'm going to put this storage account in Canada Central. And for the Replication, I am going to go with the cheapest option, which is Locally‑ redundant storage. Click on Networking, Advanced, click on Tags, Review + create. The validation passed, so I'm going to click on Create. Let's wait for the storage account to get provisioned. After about a minute or so, my storage account is ready. Let's click on Go to resource and check it out. I am going to create two blob containers in this storage account, one for the input and one for the output of Azure Stream Analytics. Let's just scroll down and under Blob service click on Containers, and click on a new Container. I am going to name this container input01, and the Public access level is going to stay private. Click on OK, and let's create a new container for the output. So click on new Container, and let's name it output01, and the Private access level is okay, so click on OK. Now that I have both input and output containers created, let's click on the input and upload our JSON file to this container. So click on Upload, and let's choose our JSON file, and let's click on Open, click on Upload, and my input file is uploaded. Now that we have the input and output figured out, let's go ahead and create a Stream Analytics job. I also bookmarked the Stream Analytics jobs, so I can simply click on it, or alternatively, I can click on Create a resource and search for stream and click on the Stream Analytics job, and, here, I have the opportunity to create a new job. Let's click on that, and let's name the job processblob‑sa‑job. I'm okay with the default Subscription, and let's put this job in the Resource group we created for the storage account. So let's select StreamAnalyticsDemo‑rg, and I'm okay with the Location for my Stream Analytics job. It is always a good idea to create your Stream Analytics job in the region as close as possible to your input and output so you don't get overcharged for transferring data between regions. Next, I need to decide which hosting environment I'm going to choose. I have the option of cloud, which is going to simply create the job in the cloud or I have the option to use Edge. In that case, my Stream Analytics job can be deployed to an IoT device. Let's click on Cloud. And finally, I need to decide on the streaming units. So the more input and output I have in my job, the more stream units I need to provision. I am okay with the three as a default now, so let's click on Create. And let's wait for the Stream Analytics job to get provisioned. Okay, after about 20 seconds, the job is ready, so let's click on Go to resource and check it out. Now that I have the job created, I need to go ahead and configure the input, output, and the conversion query. Under Job typology, click on Inputs. As you can see, I have the option to create a stream input or a reference input. We are going to talk about these inputs in details in the next module. For the time being, I am going to add a new stream input, so let's click on that. And as you can see, I have three options, Event Hub, IoT Hub, and the Blob storage. Let's click on Blob storage. Let's name our input input01, and here you have the option to specify your blob storage manually or go ahead and choose a blob storage from the same subscription you are in. I am going to go with the second option, scroll down, and I'm going to use an existing container. If I open the drop‑down, you see that I have the option to choose the input01 or output01 containers. I am going with the input01. And please keep in mind that your input alias doesn't need to be the same as the name of your container. So scroll down, and there is nothing else I want to change here, so I'll simply click on Save, and let's wait for my input to get configured. The input is ready. Now let's go ahead and configure the output. Under Job typology, click on Outputs and click on Add. And as you can see, I have a few options to choose from. I can choose an Event Hub, SQL Database, Blob storage or Data Lake Storage, Table storage, and much more. Let's click on Blob storage, and let's name our output output01, scroll down. For the container, I am going to choose the output container. Scroll down, there are two important settings that we need to be mindful of. The first one is the minimum rows. This means no output batch or output file will get created unless I have at least 10 rows in my output. The next setting is the maximum time. Here is the maximum time the job is going to wait to have the minimum rows requirement satisfied. So if I put 1 minute here, this means that the job is going to wait for 1 minute to have the minimum rows completed. After the 1 minute is up, the job is going to create an output regardless of the number of rows I already have in my output. So click on Save, and let's wait for the output to get configured as well. I have the output configured as well. And here comes the interesting part. Now we are ready to create our first conversion query. Click on the query, and as you can see, I have a query editor page. I can go ahead and create a query. I also have the option to test my query and examine my inputs. For example, let's make sure our input is selected. And as you can see, I have samples of my input available, and I can use that to form my query. I also have the option to click on the Refresh to get a new sample of the input. And please keep in mind that the data I have here is not static. The input data is constantly coming in, in my case, temperature data from a sensor. What I'm going to do is select two fields from the input, and the fields will be temperature Celsius and a sensor ID, and I'm going to write them to the output, as simple as that. So first, I need to decide which input I need to select from. In my case, I'm selecting from input number 1, so let's put that after the FROM clause, and I'm going to write my data into an output, and the output is output01. So if I leave the query as it is, Stream Analytics is going to read the input data and copy it to the output without any change. In my case, I'm only selecting two fields, so let's go ahead and change the query. So I am selecting the temperature Celsius and also the ID of my sensor. Let's save the query. Before starting the job, I have the option to test the query as well. Let's click on Test query, and as you can see, I have the output generated for me, and this is exactly what I expect. So now we are ready to start this job and see it working in action. What I'm going to do is open the storage account in a separate tab, click on my storage account, click on the containers, and as you can see, I have an input and output container. Let's go ahead and remove the file we uploaded to the input, click on OK, and I'm going to open a new instance of the browser so I can have the output container is accessible as well. Click on containers and click on the output container. And if I refresh, you see that I don't have any output file created for me and same goes with the input. So now let's go back to the first tab, click on the Overview, and start the job. When starting the job, I have three options. I can start the job now, I can start the job at a specific time, or I can have the job look at the data it missed since the last time it was stopped. In my case, I'm okay with the now option. So choosing that, and click on Start, and let's wait for the job to get started. Okay, as you can see, the job is started successfully and the status is running. So now let's take a look at the output container, refresh it, and as you can see, I don't have any output available, and this is because I don't have any input stream available to my job, so let's go ahead and upload our input JSON again. So click on Upload, click on the files, and let's choose our JSON file, and let's upload it. Let's give the job a few seconds. Now let's go ahead to the output tab and click on Refresh, and as you can see, I have the output created for me. Let's click on that, click on Download, and open the file, and here we go. If I scroll down, you can see I have exactly 50 records of the temperature information with only two fields in each record, the temperature Celsius and a sensor ID. Let's close that and let's get back to the job. This was a very simple example. We are going to build on this example in the next module demos. For the time being, you can stop this job, click on Yes, and if you don't need the job, you can go ahead and delete it. And this concludes our demo. Thanks very much.

Summary

We started this module by talking about live stream analytics concept and its use cases. There are many scenarios which you can apply live data analytics to from processing audit logs from an online banking session to monitoring temperature in a power plant. We also talked about live data processing systems. The system should have enough processing power so that inputs can be ingested without any delay. So generally, these systems cannot be deployed to commodity machines. We also talked about the technology choices you have in Microsoft Azure to implement live data processing systems. A few of these choices are Azure Functions, WebJobs, and Azure Stream Analytics. Azure Stream Analytics is the main focus of this course. After that introduction, we changed our focus to Azure Stream Analytics. This is a service offered by Microsoft Azure, and it is specifically designed to process live data streams. We talked about different kinds of inputs and outputs supported by this system. You can use Azure Blob storage, Azure Event Hubs, and Azure IoT Hub as the input of Azure Stream Analytics. There are many different types of outputs supported by Azure Stream Analytics, including Azure Blob storage, Event Hubs, Azure SQL Database, and Power BI. We also talked about the concept of windowing. Every live data processing system offers an easy way to work with events falling into a specific time window, and Azure Stream Analytics is not an exception. We talked about four different windowing functions offered by Azure Stream Analytics, the Tumbling window, Hopping window, the Sliding window, and the Session window. And finally, we saw how we can create an Azure Stream Analytic job. In the module's demo, we provisioned a new Azure Stream Analytics job. Join me in the next module where we are going to take a look at Azure Stream Analytics inputs in more details.

# Configure Azure Stream Analytics with Event Hub and Blob Storage Inputs

## Overview

Hi everyone. I am Reza Salehi. In this module, we are going to talk about the Event Hub and Blob storage Inputs for Azure Stream Analytics. We will start the module by general overview of Azure Stream Analytics inputs. We will talk about the nature of live data streams. Azure Stream Analytics supports two categories of inputs, the reference data input and the stream input. We will talk about these input categories and their use cases. Later in the module, we will talk about the supported reference inputs for Azure Stream Analytics, and these input types are the Azure Blob storage and Azure SQL Database. Later, we are going to talk about the supported stream inputs for Azure Stream Analytics. As you saw in the previous module, there are three different services that you can use as the stream inputs as your Event Hubs, IoT Hub, and the blob storage. We will spend a bit more time explaining these services and the type of inputs that go with them. And finally, we will see how these inputs can be configured with Azure Stream Analytics in the Azure portal. In the module's demo, we are going to see it all in action. We will work with both the stream inputs and the reference data input. Let's get started.

Azure Stream Analytics Stream Data Input

We have briefly reviewed Azure Stream Analytics data flow in the previous module, and you also should remember this diagram. The data flows from the left side of the diagram to the right side. Some sort of live data stream is ingested from a connected device, a sensor, or a few log files. These events in turn will be ingested by Azure Stream Analytics, processed, and then sent to any configured output. In this module, we are going to focus on the input part of this diagram. In particular, we are going to talk about Azure Stream Analytics input types. There are two types of input categories supported by Azure Stream Analytics, the data stream input and the reference data input. Let's go ahead and have a closer look into these input categories. A data stream input can be any live data stream which you need to process in real time and act upon. You need to think about the data stream as an ongoing sequence of events over time. This is not some sort of a static file that you put in the blob storage and forget about. A data stream is ingested into Azure Stream Analytics once its processed and the appropriate output is generated for that input. To be able to have new output from Azure Stream Analytics job, you need to constantly generate data stream inputs. Azure Stream Analytics support multiple inputs. You can take multiple inputs and use them in your queries. You need at least one data stream input to be able to have a working Azure Stream Analytics job. Reference data inputs are not mandatory, and you only use them if you need them. And as you mentioned a couple of times, there are three different services you can use to provide data stream inputs, Azure Event Hubs, Azure IoT Hub, and Azure Blob storage. Let's just start with Azure Event Hubs. This is a very popular service offered by Microsoft Azure. You can configure any custom application to send events to Azure Event Hubs. In fact, in the module's demo, we will see how easy it is to create a .NET application which sends new events to Azure Event Hubs. The other popular input for live data streams is Azure IoT Hub. This service is very similar to Azure Event Hubs. It's highly optimized for Internet of Things scenarios or IoTs. Using this service, you can accept events from thousands or even millions of connected devices and, in turn, send that information to Azure Stream Analytics. And finally, Azure Blob storage can be used as a stream data input, and this is what we did in the previous module's demo. This type of input is vastly used for bulk data such as log files. So let's take a look at a few examples of the data streams you can ingest using these services. Azure Event Hubs can be used to ingest events from a custom stock trading application, Azure Event Hubs or Azure Blob storage can be used to ingest events regarding online banking activities. Azure SQL Database activities are logged and also can be ingested as live data streams. Data from heat sensors in a power plant can be sent to Azure Stream Analytics using IoT Hub or Event Hubs. You can also use Azure Event Hubs to ingest data generating by a gaming engine. And finally, Azure Blob storage can be used to ingest log files from a custom application.

Azure Stream Analytics Reference Data Input

The second type of input category you can use with Azure Stream Analytics is a reference data input. Think of reference data input as some sort of data which doesn't change or changes very slowly. An example of this kind of input is metadata lookup. Azure Stream Analytics can accept reference data from two different input types, Azure Blob storage and Azure SQL Database. For example, the reference data information can be stored as a CSV or JSON file and put into Azure Blob storage or it can be saved into a table inside an Azure SQL Database. This data, in turn, can be fed into Azure Stream Analytics. So what are the use cases of reference data? If you have worked with any database, you are probably familiar with the concept of lookup tables, used for some sort of metadata for your objects inside a lookup table. That data doesn't change or changes very slowly, and you can use that table in joins with other entities inside your database. A reference data for Azure Stream Analytics plays the exact same role. You can store metadata information about the objects you are going to process in your reference data. Imagine you're accepting live events from a few devices. Each event coming from a device contains a device ID. You can store metadata for all your devices in a reference data input and then use them when processing your live data. This information can include the device capacity, name, and model. You can also use reference data for some sort of whitelisting. Imagine you are accepting events from a few devices, but you want to make sure only events from allowed devices are processed. You can have a list of registered devices and configure it as a reference data input, and each time some live data is coming in, cross check it with your trusted entities list. Other examples include acceptable thresholds, for example, allowed temperatures you want to act on or more. In general, any lookup or slow changing data can be configured as reference data inputs. Configuring extreme and reference data inputs is very easy and can be done programmatically or in Azure portal. First of all, you need to provision a new Stream Analytics job, and this is what we did in the previous module's demo. You simply create a new instance of a Stream Analytics job, wait for it to get provisioned. When in the dashboard of your Stream Analytics job, you have the option to configure inputs. Simply click on the Inputs link under Job topology. And you have the opportunity to add both the stream input and reference input. For example, clicking on Add stream input, you'll be presented with three options, Event Hub, IoT Hub, and Blob storage. You can go ahead and configure your desired input accordingly. Same goes with the reference data input. You can easily add a Blob storage reference input or SQL Database reference input. And for each of these options you choose, you need to go ahead and configure your input details accordingly. For example, in my case, I am going to use Event Hub for my data stream. I can easily configure that in the Azure portal. We are going to see this in action in the module's demo.

Demo: Sending Events to Azure Event Hubs in .NET

In this demo, we are going to set the stage for the next demo. First, we are going to provision an Event Hubs instance, and after that, we will create a small .NET application to send telemetry data to Azure Event Hubs. So first, I need to go ahead and create a new Event Hub. This Event Hub instance is going to accept live data events generated by my application and, in turn, Azure Stream Analytics can take the data in. I already bookmarked Event Hubs, so I could simply click on that, or alternatively, I can click on Create a resource, and search for event, and click on Event Hubs, and let's click on Create. Let's choose a name for the Event Hub. This name should be unique, so let's name it StreamAnalyticsDemops01. It looks like the name is available. For the pricing tier, I am going to choose Basic, which is more than enough for the purpose of this demo. Scroll down. I am okay with the default subscription, and I'm going to put this Event Hubs instance into the resource group I created for this course, so let's choose StreamAnalyticsDemo-rg. As you remember from the previous demo, my Azure Stream Analytics is created in the Canada Central region, so I'm going to create my Event Hubs in the same region as well, so let's just scroll up and click on Canada Central. The default throughput unit for my Event Hubs is 1, and this is more than enough for the purpose of this demo. You can always go ahead and enable auto inflate, so Azure is going to assign more throughput units if you have more traffic. I don't need that, so let's uncheck it and click on Create. And let's wait for my Event Hubs instance to get created. Okay, my Event Hubs instance is ready, so let's click on Go to resource and take a look, and I'm in the dashboard for my Event Hubs namespace. As we can see, the status of this Event Hubs namespace is active, so it is ready to accept events. The next thing I need to do is to create credentials so my client applications can connect to this Event Hubs namespace and send events to it. Under settings, click on Shared access policies, and I'm going to add a new one, so let's click on Add, and let's name the policy TempratureData, and I'm going to check Send and Listen permissions. Click on Create, and let's wait for the SAS policy to get configured. And my SAS policy is ready now, so if I click on that, I have primary and secondary keys, which I can use in any client application which needs to send events to this Event Hubs namespace. Let's copy one of the connection strings into the clipboard and head over to Visual Studio. I am in Visual Studio 2017, and I already created a simple console application in .NET Core. This console application is going to send events to my Azure Event Hubs namespace. To be able to work with Azure Event Hubs, I need to install a NuGet package, and I already did that, so if I right-click on the project name, and click on Manage NuGet Packages, and click on the Installed tab, as you can see, I have Microsoft Azure Event Hubs NuGet package installed. If you don't have it installed, you can easily head over to the Browse tab and install the package. After installing that package, all you need to do is to include Microsoft.Azure .EventHubs namespace into your project. So having done that, let's go ahead and take a look at the application code. As you can see, I need to fill in the EventHubConnectionString, so let's paste what I have in clipboard, and let's break this line so it's more readable. And also, I need to specify an Event Hub name, so let's go back to the Azure portal and grab an Event Hub name. I am back in the Azure portal. Let's just scroll down and under Entities, click on Event Hubs, and let's go ahead and create a new Event Hub. We have already created the Event Hubs namespace. Now we need to go ahead and create the Event Hub inside it. So clicking on Event Hub and let's name it TempratureDataHub01. I'm okay with the default partition count, so let's click on Create, and my Event Hub is created. Let's click on that, and I'm going to quickly grab the name. And let's head over to Visual Studio and paste the name in. So now I have the Event Hub namespace connection string and the Event Hub name in place. As we can see, in the main part of the application, I am creating a new instance of Event Hubs connectionStringBuilder and passing my connection string. I also need to specify the Event Hub name. After that, we are going to create a new Event Hub client using the connectionStringBuilder. And after that, simply calling your method SendMessagesToEventHub, and I'm sending 50 messages. And let's go ahead and take a look at the details of the method which is sending the messages. So let's click on the method which is sending the messages and press F12. In this method, we are creating a few random temperatures and send it to the Azure Event Hub. I am doing so because at the moment, I don't have access to any real heat sensor in a power plant, so I'm simply faking the data for the purpose of this demo. As we can see, the data will be sent in the JSON format, and each row in the data has three fields, Id, TempratureCelcius, and a SensorId. If I click on the definition of this event and press F12, you can see that my class also has three fields, Id, TempratureCelcius, and the Id of the sensor sending this information. So let's go back to our application. So as we can see, in a loop I am creating a new event, and for the temperature, I am generating a random number between 50 and 500 degrees. And for the SensorId, I'm hard coding SEN-001E. So all the data I get is only for one sensor. After that, I'll convert this message object into the JSON format added to a collection for later use and also send it. So let's uncomment the line which sends the message to Azure Event Hub. And finally, I am calling this method, SendAsync, on the Event Hub's client, which is simply sending this message to Azure Event Hub. And before going ahead and sending the next message, we are delaying for 100 milliseconds or one-tenth of a second. So let's just scroll up, Ctrl+Shift+B, so I have a clean build. So before going ahead and configuring Azure Stream Analytics to accept events from Event Hub, let's go ahead and quickly test this application to make sure it is actually sending events to Azure Event Hub. So let's go ahead and start the application. Step in, step in again. So in this step, we are creating an Azure Event Hub client from the connection string. Step in. It looks like the connection is opened successfully. And now it is time to call the method which is sending 50 messages to Azure Event Hub. So step in, step in again, and one more time. So now, I already created the event object, which I'm going to send to Azure Event Hub. Let's hover over that and open it. And as you can see, my event has an automated Id, a SensorId, and a randomly generated temperature, which is around 373 degrees Celsius. Step in one more time, and this step has serialized this object into a JSON string. If I hover over that and click on the magnifier, you can see the content of this stream, and this is the stream event we are going to send to Azure Event Hubs. So let's close that, step in one more time, and in this step, we are going to actually send the message to Azure Event Hub, so let's see if this step succeeds. Step in. It looks like the message is successfully sent, so I'm going to remove the breakpoint here and click on Continue. And as you can see, I'm sending messages at the rate of 10 messages per second to Azure Event Hubs, and execution is done. So let's exit. So now we are ready to move to Azure Stream Analytics and configure an Event Hubs input.

Demo: Configuring Event Hubs Input for Azure Stream Analytics

Now that we have an Azure Event Hubs instance ready and events being sent to it, we can go ahead and configure Azure Event Hubs input for Azure Stream Analytics. Let's get started. I am back to the Azure portal. In this demo, we are going to use the Azure Stream Analytics job we created in the previous module's demo. Let's click on Stream Analytics jobs, and as you can see, my job is already there. Let's click on the job, scroll down, and under Job topology, click on Inputs. As you can see, I already have a blob storage input from the last module's demo. We don't need that anymore, so let's go ahead and delete it, click on Yes, and the input is deleted. Let's quickly click on the Outputs as well. As you can see, I also have a blob storage output. I will need this output for this demo, so I'm going to leave it as is. Now let's go ahead and configure an Azure Event Hubs input. So clicking on the Inputs again, click on Add stream input, and click on Event Hub. For the input alias, I am going to chose tempinput01. I am going to select the Event Hub from my subscription, so I'm going to leave the default as is. And let's click on the drop-down titled Event Hub namespace. As you can see, I only have one namespace, and this is the one I need, so I'll leave it as is. And for the Event Hub name, I have the option to create a new one or use an existing one. Let's open the drop-down, and as you can see, I already have one Event Hub there, and this is the one we are sending events to, so let's choose that. And as you remember, we created a security policy for our Event Hub, so let's make sure our security policy is selected from this drop-down. The name of the security policy was TempratureData, so let's choose that. Let's just scroll down, and let's click on Event serialization format. Azure Stream Analytics supports three main input types, JSON, Avro, and CSV. And as you remember, we created our inputs in the JSON format, so I'm going to leave JSON as a default. The encoding was UTF-8, and I'm not going to touch any other default, so let's click on Save, and let's wait for our Event Hubs input to get configured. My Event Hub is configured. Now let's go ahead and take a look at the queries. Click on the query, and as you remember, this is the same query we used in the previous demo, and we are going to use the same query for the purpose of this demo as well. I am selecting my input from the tempinput01, so I need to update the name of my input here, so let's quickly do that, tempinput01, and we need to update the select clause as well, so let's quickly do that and save the query. My query is saved now. As you remember, we had the opportunity to see a sample of our inputs in this window. Let's go ahead and see that again, so make sure your input is selected and click on the Refresh, and let's wait for the new data to come in. As you can see, I have some data coming in and that's because we already sent a few messages to our Event Hub. So before going ahead and running our Stream Analytics job, let's go ahead and empty our output container. Opening Storage accounts in a separate tab, click on that, scroll down, click on the container, click on the output, and I'm going to delete the output we already have from the previous demo, so click on the three dots and click on Delete, and click on Refresh. So now we are confident any new file showing up here is related to this demo. Let's go back to Stream Analytics, click on the Overview, and Start the job, and I'm going to choose the Now option to start the job. Click on Start, and let's wait for the job to get started. Okay, after about 2 minutes, my job is running, and as you can see, the status reads Running as well. Let's click on the Output and Refresh, and as you can see, I don't have any output files created for me. Now let's go back to our application and send a few fresh events. I'm going to make a few modifications before sending the events. First of all, I need a continuous flow of events, so instead of 50 events, I am going to send 500 events, and I'm going to update the delay amount to half a second. This means we are going to send a new message, each half a second, and we are going to send 500 messages in total. This gives us a continuous flow of events for 250 seconds or about 4 minutes. So let's start the application, remove the breakpoint, and click on Continue. So now the events are coming in. Let's go back to the Azure portal, click on the first tab, click on the Query, make more room here, and as you can see, I have fresh data coming in. I can go ahead and click the Refresh button so I can get the most refreshed data in my screen. So now let's go back to the output container and see if you have any output file created for us. Click on Refresh, and here we go. We have a new output file created. Let's click on that, download it, and open it in Visual Studio Code, and as you can see, I have all fresh information generated for me by Azure Stream Analytics. So let's close that and get back to our job. So as you can see, my job is working, and I'm continuously getting new messages, so the output is going to continuously change. So if I go back and keep refreshing, you can see that the size of my output file is changing. This is because Azure Stream Analytics is appending more output at the end of this file. So let's go back to Visual Studio, stop the application. Let's go back to our job, click on the Overview, and stop the job, click on Yes. And this concludes our demo. Thanks very much.

Summary

We started this module by general overview of Azure Stream Analytics inputs. Azure Stream Analytics supports two input categories, the reference data input and live stream input. You can use Azure Blob storage and Azure SQL Database for your reference data input, and this is the data which is slow changing or doesn't change at all. A few examples are lookup tables, list of allowed devices, and list of temperature thresholds. These are the values which don't change or change very slowly, so we can configure them as reference inputs and use them when processing your live data. We also talked about the supported stream inputs, and these are Azure Event HUBs, IoT Hub, and blob storage. We also went over the steps you need to take to configure these inputs in the Azure portal. And finally, in the module's demo, we worked with Azure Stream Analytics inputs. We also developed a small .NET application that sends events to Azure Event Hubs. As you saw a couple of times, we are using queries to project the input we are taking to the output of the service. In the next module, we are going to have a deeper look into Azure Stream Analytics query language. This is an interesting module, so see you there.

Query Data Using Azure Stream Analytics

Overview

Hi everyone. I am Reza Salehi. In this module, we are going to see how to query data using Azure Stream Analytics. As you remember, we talked about Azure Stream Analytics and the supported input types. You can use Event Hubs, IoT Hub, and Azure Blob storage for a stream data input, but we still need to answer the question, what kinds of data formats are supported using these services? And this is what we are going to cover at the beginning of this module. We will also answer the $1, 000, 000 dollar question. As you remember, Azure Stream Analytics takes some streaming data in and generates an output. We are going to see how is the input mapped to the output of the service, and the answer is Azure Stream Analytics queries. These queries are used to take the input, perform some processing, and generate an output, which, in turn, will be consumed by other services. Azure Stream Analytics queries are written in Azure Stream Analytics query language. If you are familiar with T‑SQL, you already know more than 90% of this querying language. We are going to review the data types, language elements, and built‑in functions of Azure Stream Analytics query language. In the next section of the module, we are going to talk about event timestamps as well. As you remember, any piece of data coming in to Azure Stream Analytics has a timestamp, and Azure Stream Analytics is using these timestamps to perform processing on the data. Here, we are going to see which time elements can be used as timestamps for the data events. And in specific, we are going to talk about event time and arrival time. We are going to see the differences between these two time elements, and we are going to see how you can tell Azure Stream Analytics to use which of these time elements as timestamps. We will also talk about event ordering policies. These policies enable Azure Stream Analytics to deal with events, which arrive late or not in order, and we are going to have a few demos in this module. In the first demo, we are going to run a few queries written in Azure Stream Analytics query language, and we are going to take a look at the output. In the last demo of the module, we are going to see how event ordering policies can be configured in the Azure portal. This is the diagram showing how the data

Stream Analytics Supported Input Formats

flows through Azure Stream Analytics. In the previous modules, we talked about the input section of this diagram. In this module, we are going to focus on the processing section inside Azure Stream Analytics itself. Before answering that, we need to clarify Azure Stream data input formats. As you remember, we can use Azure Event Hubs, Azure IoT Hub, and Azure Blob storage to send streaming data into Azure Stream Analytics. But the big question remains. Which data formats are supported as input data streams? In other words, what kind of data formats can we send to Azure Stream Analytics using these three services? Azure Stream Analytics can accept the streaming data in CSV, JSON, and finally Avro format. I'm sure we all know about CSV and JSON, but let's take a deeper look into Avro. Apache Avro is a data serialization framework. You can use this format to send data over the wire and store it. Apache Avro was developed within Apache's Hadoop project, and Hadoop has native support for Apache Avro, so let's see how Avro packages the data. The data in Avro has two sections, the schema part and the actual data part. Apache Avro uses JSON to define the schema for the data, so it can specify what kind of data format you're transferring. Is it integer, datetime, string, etc.? The main part of the data or the data body, if you will, we will serialize into binary format. So these two sections of the Avro format are married together. So as you saw, Azure Stream Analytics supports three main data formats out of the box, CSV, JSON, and Avro, and you can see that when you are creating an input for your Azure Stream Analytics. If you open a drop‑down titled Event serialization format, you can see JSON, Avro, and CSV. You can also see the fourth option named Other. Azure Stream Analytics allows you to choose any custom data format you like as long as you go ahead and use C# to develop a deserializer for your custom data type. This means on top of JSON, Avro, and CSV, you can invent your own data format and use it to provide streaming data into Azure Stream Analytics. For the purpose of this course, we are going to use JSON as a streaming data input into Azure Stream Analytics. Now that we cleared the input formats, let's go ahead and take a look at Azure Stream Analytics query language. This language allows us to query the data coming in in the specified formats.

Stream Analytics Query Language Data Types

So far, we have figured out the input. Imagine you have a CSV or JSON input with a few fields in it. The question is how do you take in these inputs, process them, and project them to the output? And the answer to this question is Azure Stream Analytics query language. Using Azure Stream query language, you can take the input in, process it, and then project it to the output of your Azure Stream Analytics job. This query language is almost identical to T-SQL, so if you are already familiar with T-SQL, you have no problem using this query language. This is an example of an Azure Stream Analytics query language, and you can tell what this query does if you know T-SQL. First, we are selecting all the fields from the input, and the input is HeatData. I am also interested in the information coming from one specific sensor with the ID SEN-001E, so I have to use the WHERE clause to limit the information I'm going to map to the output. And finally, we are mapping the result of this query into an output with the ID output01, and we are using the INTO keyword to do so. So simply this query is going to take all the input from HeatData and only relay the information from one specific sensor to the output with the ID output01. And as mentioned, you already know this language if you are familiar with T-SQL. Let's have a closer look into Azure Stream Analytics query language. We are going to talk about data types, for example, nvarchar, bit, float, or more. We will talk about language elements such as select, where, join. And finally, we will take a look at built- in functions such as average, count, max, and others. Let's start with data types. Azure Stream Analytics query language supports a few data types, which you can use in your query language. You can use the float and bigint data types. The bit data type, which is simply true or false can be used as well. The other data type is nvarchar, which is identical to the data type you saw in T-SQL, so any type of string can be casted to this data type. Datetime is another data type you can use in Azure Stream query language. And finally, we have record and array. Array is pretty self explanatory, so we can have a lease of any supported types in an array. For example, you can have an array of datetimes or array of bits, floats, or nvarchars. Record is a key value pair, and a key and value should be supported data types. For example, you can have a key of the type float and the value of type nvarchar. This is a sample input we used in the previous module demos. As you can see, these are sample heat information coming from a heat sensor. In each record I have five fields, Id, TempratureCelcius, SensorId, CoreId, and EventTime. The Id, SensorId, and CoreId are of type string. The TempratureCelcius is of type float, and finally, EventTime is a datetime. So as you saw, Azure Stream Analytics query language is a strongly typed language. Talking about strongly typed languages, the first thing comes to mind is type conversions or casting. There are a few type casting functions you can use to convert different types to each other. The first one is CAST, the second one TRY\_CAST, and finally, GetType. You can use CAST and TRY\_CAST to convert one data type to another. GetType can be used to get the data type of a specific value. So what happens if an error occurs when you are casting some value to target data type. So type casting errors can happen in your query when you are reading the input or you are writing the output. First of all, let's take a look at this example. I am selecting DeviceId, Model, and Name from an input, and in my condition, I am taking the value of DeviceId, and I'm casting it to bigint, and I filter my input if the casted value of DeviceId is greater than 1002. So this query works perfectly if this casting is successful. If it's not successful, we are going to deal with type casting errors. Type conversion errors can happen when you are reading the input data. In this case, the job is going to drop the event, as simple as that. If the type conversion error is happening when you are outputting the data in your query, this error will be handled by the error policy. You can configure the error policy to drop or retry the work. You can configure the error policy in the Azure portal. Simply navigate to the Azure Stream Analytics job page, scroll down, and under Configure, click on Error policy. Under Output data error handling, you have the option to decide what should happen if an error happens when writing the output. You can drop that specific record or you can retry. We are going to take a look at this configuration in the module's demo.

Stream Analytics Query Language Elements and Functions

There are a few Azure Stream Analytics query languages which you need to be aware of. And, again, if you are familiar with T-SQL, you'll have no problem understanding these language elements. A few examples are the select keyword, from, over, where, union, and into and the usage of these keywords is exactly the same as T-SQL. For example, select can be used to choose a few fields from input. The where keyword can be used to put some predicate to select the specific values. The same goes with union and into. The other examples are group by, having, join, with, case, and coalesce. As you saw before, the into keyword can be used to specify an output stream so it can send the output of your processing to an output service. The other keyword is apply, which can be used to invoke functions for each and every data row. And finally, create table can be used to define the schema for the input payloads so it can have strongly typed inputs. So how can we get more information about these keywords? Microsoft did a great job in creating documentation for Azure Stream Analytics language elements. I have added the link to this documentation in the module files, so if you navigate to this documentation page, you will see detailed explanation of all these query language elements with a few examples, so make sure you take a look before going ahead and writing your own queries. And last, but not least, let's talk about query language built-in functions. As the name suggests, built-in functions are the functions created by the platform so we can use them to perform some useful actions on your data. I'm sure you've used built-in functions before in SQL Server or MySQL or other languages. The same concept applies here. Up to the time we are recording this course, there are 11 categories for query language built-in functions, so let's go ahead and take a look at them quickly. The first category is aggregate functions. These are the functions that take a group of values in and return a single value. A few examples are average, maximum, minimum, and much more. Analytic functions are a group of functions that take a group of values in and make a decision, for example, detecting anomalies. Array functions can be used to perform some actions on arrays, for example, giving the length of an array. GeoSpatial functions can be used to perform real-time actions for GeoSpatial data, for example, giving the distance between two points. The other category is input metadata functions and record functions. As you remember, we talked about the record data type previously in this module. Record functions can be used to perform actions on the record data type, for example, you can use these record functions to get the values of records. Another category of built-in functions are the windowing functions, and we talked about these windowing functions previously in this course. We also covered the conversion functions. A few examples are CAST and TRY\_CAST. These functions can be used to convert a source data type to the destination data type. Date and time functions are another category of built-in functions, and as the name suggests, they can be used to perform actions on date and time. You can also have access to the mathematical functions. An example would be creating a square of a value. And finally, we have string functions, which can be used to perform actions on the strings, for example, get linked or concatenate two strings. And, again, Microsoft did a great job in documenting Azure Stream Analytics query language built-in functions. I have added the link to this documentation in the module files. If we navigate to the built-in functions documentation page, you can see all the function categories we talked about. You can click on any of these categories, in our case, Aggregate Functions, and you will be redirected to a page listing all aggregate functions. You click on the function you are interested in, for example, average, or AVG, and you'll get detailed explanation of the function along with a few examples. So, again, make sure you take a look at this great documentation before going ahead and writing your queries. We are going to use a few of these functions in the upcoming module demo.

Stream Analytics Timing and Timestamp By

So before going ahead and seeing some of these queries in action, we need to talk about Stream Analytics timing. There are two concepts we need to clarify regarding Stream Analytics timing. The first one is the event timestamp and the difference between arrival time and event time. We also need to talk about event ordering policies. As we have mentioned a couple of times throughout this course, every piece of data coming in to Azure Stream Analytics has a timestamp, and Azure Stream Analytics is using this timestamp to process the data over time. So what is the value of this timestamp? Is it the actual time the event was generated or is it the time the event was ingested into Azure Stream Analytics? To answer this question, we need to first talk about event arrival time and the event time. Imagine you have a heat sensor in a power plant. This sensor created a heat event at exactly 1 hour and 10 minutes AM, so you have the heat information for exactly that time. This means the event time is 1 hour 10 minutes AM; however, to be able to use this event in Azure Stream Analytics you need to ingest it in a supported service, for example, event hops. Let's imagine this event was ingested into Azure Event Hub at 1 hour, 10 minutes, and 2 seconds AM. This is the time this event was received in the Event Hub, and this 2 second delay might be because of network latency or other issues. This means the event arrival time is 1 hour, 10 minutes, and 2 seconds. And as you can see, I have two time events to work with, so the question is which of these times is used by Azure Stream Analytics as the timestamp? The answer is the arrival time. So Azure Stream Analytics uses arrival time as the timestamp by default, so the time of my event will be 1 hour, 10 minutes, and 2 seconds AM. So let's see what is the arrival time for every single input type? For Azure Event Hub and IoT Hub, the arrival time of an event is the time the event was received by these services. In case of Azure Blob storage input, the arrival time is the last modified time of the blob object. And as you saw, by default, Azure Stream Analytics uses the arrival time as the timestamp for your event. In many cases, you might need to use event time instead of the arrival time. In our example, you might need to use 1 hour and 10 minutes AM sharp as the timestamp of your event. The question is, is there a way to tell Azure Steam Analytics to use event time instead of arrival time. The answer is yes. You can use a TIMESTAMP BY clause to do so. So using the TIMESTAMP BY clause, you can specify custom timestamp values and these custom values can be anything, including the fields in the actual event itself. Let's take a look at the sample query. As you can see, I'm ingesting an event and this event has at least three fields, the AlertTime, Temprature, and the ValveNumber. So I am selecting these three fields from the input, but I'm also adding the TIMESTAMP BY clause followed by the AlertTime field. This instructs Azure Stream Analytics to use the custom AlertTime as the timestamp for this event. So as you can see, you can use custom time for your event by simply adding the TIMESTAMP BY clause after your FROM clause. And after that, all the other timing calculations, including the windowing functions, are going to use the other time field instead of the default arrival time of the event. Using custom values for the timestamp might cause a few issues, for example, you need to be able to deal with out of order or late messages.

Event Ordering Policies

So as you know, Azure Stream Analytics is processing live data. There is a possibility that Azure Stream Analytics receives late arrivals or out of order events. One of the reasons would be using TIMESTAMP BY. Using this clause, you can specify custom time values for the timestamp. These custom values might come from devices with their skewed clocks. You might also have multiple inputs, for example, multiple heat sensors with time mismatches in their internal clocks. This is going to result some out of order messages coming into Azure Stream Analytics. Another reason for receiving late arrival messages is network latency, which is very common. So how do you deal with these messages? The answer is using event ordering policies. The first event ordering policy you can configure is late arrival policy. You need to configure a policy window in term of time, for example, let's say I have configured late arrival policy window for 5 seconds. Every message coming in has a timestamp, so the timestamp will be compared with the current time. If the message is older than 5 seconds, it can be adjusted or dropped. If the message is newer than 5 seconds, it will be accepted automatically. The other ordering policy you can configure is out of order policy. In this case, you also need to specify a policy window, let's say 1 minute. So if you have messages coming in and newer messages are already received, Azure Stream Analytics is going to compare the out of order message timestamp with the out of order policy window time. If older, the message will be dropped or its timestamp will be adjusted to the current time. If the out of order time difference is less than the out of order policy window, the message will be accepted. So how do we configure late arrival policy and out of order policies? You can easily configure them in Azure portal. Simply navigate to Azure Stream Analytics job page, scroll down, and under Configure, click on Event ordering. And as you can see, you have the opportunity to adjust the time window for both late arrival time and out of order time. You also have the option to tell Azure Stream Analytics what to do with the messages falling out of these windows. The timestamp of these messages can be adjusted to the current time or the message will be dropped and ignored. We are going to take a look at this page in the module's demo. And also one important note to remember, event ordering policies are applied only if TIMESTAMP BY is used in your queries. If you are not using TIMESTAMP BY, the arrival time of the event will be used, and there is no need for event ordering policies to kick in. Now let's go ahead and take a look at a few queries in action.

Demo: Stream Analytics Query Language

Okay, time for a demo now. In this demo, we are going to see a few Azure Stream Analytics queries in action. We are going to take a look at data types, joins, and a few aggregate functions. We will also see how to configure event ordering policies in the Azure portal. As you remember, in the previous module demos, I use this small console application to generate a few events, send them to Event Hub, and in turn get them processed in Azure Stream Analytics. In this demo, we are going to build on the previous module demos. First of all, I would like to go ahead and add a reference input to my Azure Stream Analytics job. Let's click on the Program.cs file. And as you remember, we use the SensorId SEN‑001E. I would like to add a reference file, which has some metadata about all the sensors I am using in my application, so I can go ahead and use the sensor details in my Azure Stream Analytics job output. This is the JSON file I created, which holds some metadata about my sensor. Because I am using only one sensor in this example, I only created one record in this JSON file; however, I could always go ahead and add more items to this JSON array. As you can see, the SensorId here is identical to the SensorId I used in my event generator, and I have added two metadata fields to this sensor, the Brand and the Model. I have named the file seninfo.json. Let's go back to the Azure portal and quickly configure this. I am in the Azure portal, and I have already uploaded the sensor file to a storage account container. Let's go ahead and verify that. I'm clicking on the Storage accounts, click on inputaccount01. Let's just scroll down under Blob service. I'm clicking on Containers. And as you can see, I created a new container and named it sensorreference. If I click on that, you can see my file uploaded there. I can click on that, and download the file, and open it. And as you can see, this is the exact same file I showed you. Now let's go ahead and add this as a reference input. I'm clicking on Stream Analytics jobs, and I click on my job. Under Job topology, let's click on Inputs. And here, instead of clicking on Add stream input, I'm clicking on Add reference input and choose Blob storage. Let's name this input sensormeta, which is short for sensor metadata. Let's just scroll down and make sure our container is selected in the container drop‑down. I am only having one file in that container, so I don't need to specify a path pattern. Let's just scroll down, and nothing else to change, and let's click on Save. And let's wait for this new reference input to get configured. Okay, my reference input is configured. Let's click on Query. And as you can see, this new reference data is showing up under Inputs. If I click on that, I should be able to see a preview of my inputs. And here we go. As you can see, I'm seeing the SensorId, Brand, and Model as expected. So now I have two inputs to work with, the HeatData input, which is accepting input data from my .NET application, and this is a streaming input data and also the sensormeta data input, which is a reference data. So now let's see how can we go ahead and use this reference input to get some metadata about our sensor. So I'm going to change the existing query here. I'm going to select all the fields into output01, which is my blob storage output. And I'm going to select from HeatData. Let's test this query by clicking on the Test query. And as you can see, I have the data there. Now let's go ahead and select only a few specific fields from the HeatData. So I am interested in TempratureCelcius and SensorId. Let's click on Test query and make sure we get two columns back, and here we go. So now let's go ahead and query our sensormeta data information as well. To do so, we need to use the join keyword, so basically I'm joining the data from the reference input and the streaming input. So first, let's choose an alias for my HeatData, and I choose HD, and I'm going to add a JOIN and specify sensormeta input, and let's choose an alias for it as well, so sm, and we need to add a join condition. So the join condition is sm.SensorId=hd.SensorId. And the last step, we need to go ahead and make sure we add the alias to our SELECT statement. So I'm choosing the TempratureCelcius from the HeatData, and I'm choosing the SensorId from the HeatData, but let's go ahead and choose the Brand and Model from the reference data, so sm.Brand and sm.Model. Let's click on Test query. I have the join data back. Here, I'm getting the TempratureCelcius, the SensorId, but I'm also getting the Brand and the Model from my reference data, and this is how you use the reference data with the JOIN keyword. To keep the example simple, let's go ahead and remove the join because we don't need it anymore, and I'm going to remove the SELECT from the metadata, make sure my query works by clicking on Test query, and here we go. Let's go ahead and try a few more queries. First, let's make more room here. And next, I'm going to try the CAST function. We are getting the TempratureCelcius as a float number. I really don't need all those floating points, so I'm going to cast that TempratureCelcius into bigint. Let's go ahead and quickly try that. So I'm going to cast the TempratureCelcius as bigint, and I'm going to choose a new alias for this bigint number, and let's name it Temp, and let's click on Test query. And as you can see, I'm seeing integers, not floats anymore. And please note that by clicking on Test query, we are using some cached values from the input. These are not live streaming inputs. And as you can see, my application sending the input as live stream is not started yet. So now let's go ahead and take a look at some windowing functions. Imagine I would like to get an average of this temperature every 5 seconds. Let's go ahead and see how to do it. First, I'm going to remove this CAST because we don't need it anymore, and let's make our query a bit simpler. So the first thing I would like to do is go ahead and add a GROUP BY clause. So after GROUP BY, I need to use a windowing function. I am going to get the average every 5 seconds, so I have a fixed size window. So let's add TumblingWindow after the GROUP BY. And as you remember, I need to add the unit of time and the number, which is 5 seconds. Because I added GROUP BY, I need to go ahead and change my SELECT by adding an aggregate function. And as you remember, we need to get the average of the TempratureCelcius, so let's go ahead and grab the average of TempratureCelcius. And finally, we need to add our SensorId to the GROUP BY as well. So now our query is ready. Let's click on Test query and see if we get any information back. And here we go. I got an average back. If I run this query with live data, I am going to get this average every 5 seconds, and we are going to see that shortly. First, let's go ahead and fix this average so we get an integer back. So let's cast my average as bigint, and let's name it AverageTemp. Let's test the query again, and here we go. So now let's go ahead and see this job in action. For this demo, I am going to create an Azure SQL Database output so we can go ahead and see the output in the database. As you see, I have two columns here, the averagetemp and the sensorid. I need to go ahead and create a table with the exact same columns in Azure SQL Database, so let's go back to Azure SQL Database and take a look. I am in Azure SQL Database, and I already created a table and named it ProcessedHeat. It has two columns, Temp and SensorId. Let's go ahead and change our query so the name we have here matches the name of the column for the temperature, and let's save the query. Going back to Azure SQL Database, and let's grab the table name, and now we are ready to create an Azure SQL Database output. I am in my Azure Stream Analytics job page. Let's click on Outputs, and as you can see, I already have a blob storage output, which we are not going to use for this demo. I'm clicking on Add and click on SQL Database. For the Output alias, I am going to choose sqloutput and make sure our database is selected, which is correct here. I'm going to quickly put the username and password for my Azure SQL database here. And as you remember, the table name is ProcessedHeat. Let's click on Save, and let's make sure the output is configured correctly. Okay, as you can see, my output was successful. Before proceeding, there is one note that you should be aware of. Let me click on Azure SQL databases. Let's click on my database, and let's click on Set server firewall. You need to make sure that Allow Azure services and resources to access this server option is on. If this option is off, Azure Server Analytics job is not able to write the data to Azure SQL Database. So let's close this. Now that we have the output configured, let's go back to the query and update our query. So here, instead of writing to output01, I would like to write to sqloutput. Let's save the query. Let's make more room here. Let's test the query for the last time. And as you can see, I get the average back, so now we are ready to start our job. Click on Overview, and let's click on Start, Start, and I'm going to go back to Visual Studio and start my application as well, so it is going to start sending events as well. And let's go back to the Azure portal and wait for the job to start running. Okay, looks like the job is running now. So if we are lucky, we should get some output in Azure SQL Database. Let's wait for a few seconds, and I'm going to select everything from the ProcessedHeat table. And as we can see, I'm getting some averages back. So hopefully, every 5 seconds, I'm getting a new row in here. And that's right. I just got a new row. And if I wait for 5 seconds and execute the query again, I'm getting a new average. So as you can see, my job is working perfectly. So if for some reason you don't see any output in your Azure SQL Database, you need to take a look at the activity log. Activity log is your best friend for troubleshooting Azure Stream Analytic job. For example, if I click on the Activity log and I scroll down, you can see all the details about my job executions. So if some error is happening, you will be able to find the reason here. For example, as you can see, about 54 minutes ago we had a failed job. If I open that, scroll down again, and click on the error, and click on the JSON field, and scroll down, you can see this job failed because our table didn't have the right column names so the job couldn't write the output to the table. Let's click on Overview, Stop the job, click on Yes. Okay, and the job is stopped, and this concludes our demo. Please make sure you clean up all the resources you created for the purpose of this demo if you don't need them anymore. This includes the new database we created for this demo, the storage accounts, and the Stream Analytic job. In this demo, we are going to configure the policies related to Azure Stream Analytic job. So as you can see, I am in the overview page of my Stream Analytics job, so let's just scroll down, and under Configure, let's click on Error policy. Here we can specify what should happen in case of errors when Azure Stream Analytics job is trying to write an output. The default is Retry, so Azure Stream Analytics tries to retry the job in case in failure or you can go ahead and choose Drop. I am choosing Drop, and saving, and confirming. So from now on, if any error happens including casting events when the job is writing an output, that specific record will be skipped. Now let's go ahead and take a look at event ordering policies. Under Configure, click on Event ordering. And as you can see, I have the option to specify the windows for both out of order and late arrived events. The default window for the late arrived events is 5 seconds, and the default window for out of order events is 59 minutes or close to an hour. And as you can see, for any event arriving outside these windows, we have two options. We can accept the event and adjust its timestamp to the current time or we can simply drop the event. Let's leave it as Adjust and click on Save, click on Yes, and this concludes our demo. Thanks very much.

Summary

We started this module by talking about supported input formats for Azure Stream Analytics. As you saw, Azure Stream Analytics supports CSV, JSON, and Avro out of the box; however, you can go ahead and develop your custom serializer and deserializer, and after that, you can use any custom data type for Azure Stream Analytics input. After that, we answered this important question, how is the input mapped to the output in Azure Stream Analytics? And the answer is, Azure Stream Analytics query language. We talked about data types, language elements, and built-in functions of Azure Stream Analytics query language. If you are familiar with T-SQL, you'll feel at home here, and you already know Azure Stream Analytics query language. We also talked about data stream event timestamps. As we mentioned several times, every piece of data coming into Azure Stream Analytics has a timestamp, and Azure Stream Analytics uses these timestamps to process the data. We also answered the question, which data event time will be used for timestamps by default? Is it the event time or the arrival time? And the answer is the arrival time; however, you can use the TIMESTAMP BY clause to use any custom time, including the event time for your input data. We also talked about event ordering policies for late arrivals and out of order events, and these event ordering policy windows can be configured from within the Azure portal. In the module's demo, we first saw a few Azure Stream Analytics queries in action, and after that, we configured event ordering policies. Are you familiar with Microsoft Power BI? Have you used it before? Join me in the next module if you want to see how Azure Stream Analytics output can be sent to Microsoft Power BI so the Azure Stream Analytics output can be easily visualized. Thanks very much, and see you in the next module.

Implement Azure Stream Analytics Data Visualization with PowerBI

Overview

Hi everyone. I am Reza Salehi. In this module, we are going to talk about Azure Stream Analytics and Power BI integration. As you know from previous modules, Azure Stream Analytics supports several outputs. In this module, we are going to talk about two of them in specific, Microsoft Power BI and Azure SQL Database. You can use Microsoft Power BI to visualize the output of Azure Stream Analytics. The same output can also be saved into Azure SQL Database for later analysis. We will also answer the question on how can you use Power BI with Azure Stream Analytics? You should already have an account with Power BI to be able to integrate Azure Stream Analytics with it. We will also give a brief introduction on Microsoft Power BI; however, this course is not about Microsoft Power BI. I am going to provide a few resources, including a great Pluralsight course if you'd like to know more about Microsoft Power BI. Before jumping into the demo, I am going to show you how to configure Power BI output for Azure Stream Analytics. We are going to do that in the Azure portal. And finally, we are going to have two demos in this module. We are going to visualize the output we got in the previous demos using Power BI. We will also save Azure Stream Analytics output into Azure SQL Database and configure Power BI to read this data from the SQL Database. This is an interesting module, so let's get started.

Azure Stream Analytics Power BI Output

I'm sure you'll remember this diagram from the previous modules. This diagram shows how the data flows into Azure Stream Analytics. In the previous modules, we talked about the left side of this diagram. We talked about a stream and referenced data input. In the previous module, we also talked about Azure Stream Analytics query language. Using this query language, you can take the input, process it, and then send it to the output. This module is dedicated to the right side of this diagram. And specifically, we are going to talk about two outputs of Azure Stream Analytics, the Power BI, and Azure SQL Database output. So what is Power BI? Power BI is a business analytics solution that lets you visualize your data. You can use advanced charts in Power BI to easily visualize your data. Power BI also allows you to create reports from your data. These charts and reports can be easily embedded into your applications using Power BI widgets. So the question is, why would you want to use Power BI with Azure Stream Analytics? And the answer is simple. The nature of the data flowing into Azure Stream Analytics is live data. Using Power BI, you can have real-time visualization and insights into your data, and that can be really valuable. Finding patterns in data is much easier when you're looking at the data. So what is the next step if you would like to use Power BI with Azure Stream Analytics? You need to visit the Power BI website. In this website, you can find all the information you need on Power BI, technical documentation, and pricing. You also have the option to download the Desktop version of the Power BI. This is a free download; however, before being able to use Power BI, you need to buy an account. A Power BI trial account can be used as well. There are a few pricing options for Power BI, but the option I chose is the Power BI Pro, and it costs about $10 dollars a month. I've already purchased a Power BI Pro account, and I'll be using it in the module's demo. So to recap, a Power BI account is needed to set up Azure Stream Analytics output. So we have the Power BI account handy. What is the next step? Now you need to go ahead and set up a Power BI output. Simply navigate to the dashboard page of your Azure Stream Analytics job. Under Job topology, click on Outputs. Here, you have the option to add new outputs by clicking on the Add button. Click on the Add button and choose Power BI from the options in the list. After that, you have the opportunity to specify the details of your Power BI output, and after that, the output is ready to be used. We are going to go through this exercise in the module's demo. So this course is not about Power BI, so I am assuming you have a basic understanding of Power BI before jumping into the demo. There are a few great courses about Power BI in the Pluralsight library, and I'm specifically recommending this course titled Getting Started with Power BI by Stacia Varga. There are other great courses on the same topic from the same author as well, so I would strongly recommend you check these courses out if you are going to use Power BI with Azure Stream Analytics. So now let's go ahead and take a look at Azure Stream Analytics Power BI output in action.

Demo: Azure Stream Analytics Power BI Output

Time for the first demo of the module. In this demo, we are going to visualize Azure Stream Analytics output with Power BI. You need to have a Power BI account or trial account to be able to follow these steps in this demo. Okay, I am logged into the Azure portal. We used this query in the previous modules to get the average of all the temperatures received from the heat sensor every 5 seconds. Let's go ahead and take a look at this query. First, I'm going to make more room here, so let's collapse the Stream Analytics jobs, and that's much better. So as you can see, I am selecting the average of temperatures in Celsius and I cast it to bigint. I'm also selecting the ID of the heat sensor, which is the source of these temperatures. We are also using a Tumbling window with a length of 5 seconds. This means every 5 seconds, I'm going to get the average of all the temperatures received in the previous 5 seconds. Let's go ahead and test this query, so click on Test query. And Azure Stream Analytics is using some cached input to give me some test results. And as you can see, I get some results back. Let's go ahead and add a time element to this result set. This will help us when visualizing this result set. So I am going to also select the System.timestamp, and I'm going to name it t. Let's test the query again, and as you can see, I have a new field with the current timestamp at the end of each Tumbling window. So now, everything is ready for us to go ahead and visualize the output of this Azure Stream Analytics job. I'm going to save the query first, and now we are ready to focus our attention to the Power BI part. So as you remember from the presentation, you need to have a Power BI account ready to be able to use it with Azure Stream Analytics. You can go ahead and purchase a Power BI license or you can use the trial version for the purpose of this demo. I am going to add the link to this website in the module files if you want to go ahead and take a deeper look. So here, I am in the Power BI website. I scroll down, and under Share with Power BI Pro, I'm going to click on the Start free trial. Here, you can go ahead and enter your work email address and set up a trial account. Please remember that you cannot use personal email addresses from Gmail, Outlook, or similar services. This email address should be a work email address. I already went ahead and created a Power BI account using an email address from my custom domain, so I'm going to close this page, and click on Sign In. And I'm going to log in with the Power BI account I have. The email address for that account is reza@ salehi.ca. Click Next, and I have registered my domain and also the email address through GoDaddy. That's why I'm redirected to the GoDaddy page. So I'm going to quickly enter my password, click on Sign In, save the password, and I'm going to stay signed in. And here we go. I am in my Power BI dashboard. As you can see, I have a workspace and a few reports in it. For the purpose of this demo, I am going to go ahead and create everything from scratch. So first, I am going to go ahead and delete the existing reports that I have. So click on My workspace, click on Reports, and I'm going to go ahead and delete all the existing reports that I have. Click on Datasets, and I'm going to delete these existing datasets as well. We are going to create a new dataset momentarily. Delete. So now all I have is an empty workspace. So let's go back to the Azure portal and set up a Power BI output. I am back to the Azure portal. Let's expand the Stream Analytics job menu. I'm going to click on Outputs, and click on Add. In this list, I'm going to scroll down and click on Power BI. And let's name our output pbi, which stands for Power BI output01. And here, I cannot see any exiting workspaces. That's because I haven't connected my Power BI account with the Azure portal. Let's go ahead and do that. Under Authorize connection, click on the Authorize button, and here I need to go ahead and log in with my Power BI account. So let's log in again, reza@ salehi.ca, click on Next. Let's enter the password again. Click on Sign In. And as you can see, my account is connected, and I can see my only Power BI workspace. If you have multiple workspaces, you will see all of them in this drop-down, and you can choose which workspace to use with this Power BI output. So let's leave the Group workspace as My workspace, and let's create a dataset name. This name will be used by Azure Stream Analytics to create a new dataset in my workspace. So let's name it dataset01, and I'm going to name the table heatdata, and click on Save. And let's wait for the Power BI output to get configured. Okay, my Power BI output is ready to be used. Let's click back on the query, and we need to change our query to select the output into the Power BI one. So I'm going to replace the SQL output with pbioutput01. I'm going to save the query. Let's make more room here, and let's click on Test query to make sure the query works as expected, and here we go. So let's recap what we have so far. We have a query which selects the average of temperatures, the name of the sensor, and the current timestamp at the end of each Tumbling window, and it is sending this information to the Power BI output I created. So let's click on Overview, and to start this job, click on Start, and let's wait for the job to get started. And after about 1 minute, the job is running now. The last thing we need to do is going ahead and start sending live events. So let's go back to Visual Studio, click on Program.cs, and start the application. So now we are sending live events to Azure Stream Analytics. Azure Stream Analytics is processing these events, and it is sending the output to Power BI. So I'm back to the Power BI dashboard. And as you can see, I have a new dataset named sadataset01 under my Datasets. Let's click on the Create report button. And as you can see, I have all the fields in my output available to be used in the reports. So here I can go ahead and create any report or chart I'm interested in. Let's go ahead and add a line chart to my dashboard. Let's click on Line chart, and let's go ahead and add our data to this chart. So for the axes, I am going to use the time and the temperature. So let's drag the time and the temperature under the axis. For the Legend, I am going to use the SensorId. So in case I have multiple SensorIds, I'll get different colors in my data. And for the values, I am interested in the temperature values. And here we go. So as you can see, I have all the temperature averages showing up in my chart. I can go ahead and save this report so it can be used later. Let's name it report-heat-01, and let's save this report. So as you can see, it is extremely easy to use the data coming from Azure Stream Analytics as a Power BI output. I also have the option to go ahead and publish this report to the web. Click on File, click on Publish to web, and here you have the option to create an embedded quote. Let's click on that, click on Publish. So I have two options here. I am given a link, which I can provide in the emails, so anyone clicking on the link will be directed to this live report. I also have the option to grab the HTML code given to me and paste it an HTML page. Let's go ahead and try the direct link to this report. So I'm copying it. Let's open a new tab, paste the link, and Enter. And as you can see, I am redirected to this live page. So if I go ahead and refresh this report later in case I have new data, I'll see all the live data showing up here. This is a very powerful feature. And this concludes our demo.

Demo: Azure Stream Analytics Power BI Output with SQL Database

Okay, in this demo, we are going to configure Azure SQL Database output for Azure Stream Analytics, then we are going to configure Microsoft Power BI to use the data from Azure SQL Database to create a report. Let's get started. I am back to the Azure portal. In the previous demo, we sent the Azure Stream Analytics output directly to the Power BI. In this demo, I am going to send the same output to Azure SQL Database, and, in turn, we are going to configure Power BI to use the data from Azure SQL Database to generate the same report, so let's go ahead and do so. As you remember, we configured an Azure SQL Database output in the previous modules. Let's click on Outputs, and as you can see, I have a SQL output. Let's click on that. The database name is outputDB, and I already specified the table and username password. Let's quickly switch to Azure SQL Database and take a look at this table. I'm going to log into my database hosted in Azure SQL Database. So let's connect, click on Databases, outputDB, Tables, and as you can see, I have a table here. Let's right-click and select the first 1000 rows from this table. I already changed this table and added a new field of type datetime. This field is going to keep the third element I am selecting in my query output. Let's go back to Azure SQL Database, click on Query, and as you can see, I need to go ahead and change the output of my query to be sqloutput again. So let's click on Overview, Stop the job, click on Yes, and we need to wait for the job to stop before being able to change the query. Okay, after about 40 seconds, the job is stopped, so let's click on the Query and quickly change that. So I'm going to project my output into the sqloutput one. Click on Save query. And before starting the job, let's go ahead and empty this table. I already used this table for some testing, so I'm going to go ahead and empty the table so we can see all fresh data in it. So I'm going to delete all the records in this table, and let's go ahead and select everything. So as you can see, this table is empty now. So now everything is ready for us to go ahead and start the job now. And just to recap, after starting the job and sending live data, the output will be sent to Azure SQL Database. So click on the Overview, click on Start, and let's just start the job, and let's wait for the job to get started. Okay, the job is started now. Let's go back to Visual Studio and start our application again. So now, fresh events will be sent to Azure Stream Analytics. Let's go back to Azure SQL Database, wait for a few seconds, and I'm going to select everything from the ProcessedHeat table. Hopefully we'll have some records, and here we go. So let's wait for a few seconds so we have enough rows in this table. So far, I have about 15 records in this table, and that's enough for the purpose of this demo. I am going to leave the job running, so we will have new rows coming in. So now we need to go back to Power BI and create a report from this database. To be able to create a report from Azure SQL Database, you need to download and install the Desktop version of the Power BI. After creating the report, you can publish it to the cloud. I have already downloaded and installed Power BI Desktop, and I will add a link to the module files so you can do the same. The first thing I need to do is go ahead and sign in using the account I created in the Power BI, so let's click on Sign In. I am going to use my account email, and let's click on Enter, and let's go ahead and put in my password. Click on Sign In, and let's minimize this. So now, I am ready to create my first report in the Power BI Desktop. The first thing I need to do is go ahead and load my dataset from Azure SQL Database, so I'm clicking on Get Data, click on More, click on Azure here, and I'm going to select Azure SQL database. Click on Connect. And here, I need to go ahead and connect to the Azure SQL Database I created. We need to grab the server name and the database name. So let's quickly go back to the Azure portal, open SQL databases in a separate tab, click on my database, and let's grab the server name, and paste it here. Let's click on OK. I have already put in my database password, and that's why I can't see the databases within this server. Let's expand our database and check the ProcessedHeat table. And as you can see, I have a preview of all the data in this table. Let's click on Load, and let's wait for the refreshed version of the data to come in, and here we go. As you can see, I have all the data and all the fields within that dataset loaded. So let's go ahead and create the same report, but this time from Azure SQL Database. I'm clicking on Line chart. Let's expand that. For the axis, I'm going to drag the time and the temperature. For the legend, I'm dragging the SensorId, and for the values, I am interested in the value of the temperature. Let's minimize the filters, and I need to go ahead and make some adjustments. Click on the model icon. Click on the time. And here, I am going to change the Data type to Time and the Data time format to include seconds. Click on Temp, and everything looks good here, so let's go back to our report. And as you can see, I get the same report I got from the cloud version. So now I can go ahead and save this report. Click on Save. I'm going to name this r02, click on Save, and now I can publish this record to the Power BI cloud as well. Let's click on Publish, and I'm going to publish it to the only workspace I have. So select that, and this report is published to my Power BI in the cloud, so I can go ahead, open that report in the Power BI cloud, and share it through web or email. So let's minimize that. So before concluding the demo, I am going to go back to the Azure portal and make sure I clean up all the resources I created for this demo, including the database and the Azure Stream Analytics job. Thanks very much.

Summary

We started this module by talking about Azure Stream Analytics outputs. We talked about the Microsoft Power BI output and the Azure SQL Database output. I also gave a brief introduction on Microsoft Power BI, which is a great product by Microsoft allowing you to use advanced charts to visualize your data. You can also use Power BI to create reports on your data. This course was not about Power BI, so I have introduced a few courses in the Pluralsight library so you can go ahead and learn more about this great tool. Before jumping into the demo, I showed you how to configure the Power BI output for Azure Stream Analytics. This can be done in the Azure portal. We also had two demos in this module. In the first demo we saw how to visualize a Stream Analytics output using Power BI. And in the second demo, we configured Azure SQL Database output for our Azure Stream Analytics job so the output can be stored for later analysis. This summary concludes the module and also the course. We started this course by talking about Azure Stream Analytics inputs. We also talked about Azure Stream Analytics query language, which is used to process the input and map it to the output of the service. Now you have all the information you need to go ahead and build your own live data streaming pipelines in Microsoft Azure. Thanks very much, and see you in the next courses.