Eventhub Access Key URL: Endpoint=sb://sqlbitsdemo.servicebus.windows.net/;SharedAccessKeyName=RTAusers;SharedAccessKey=blYy3bfQJfRuu8pSftOh85MIxpCZlfcGq+AEhGde/fg=;EntityPath=bitsdemo

Access key: blYy3bfQJfRuu8pSftOh85MIxpCZlfcGq+AEhGde/fg=

Key Name:  RTAusers

RTAuser1@arrabiagmail.onmicrosoft.com

RTAuser2@arrabiagmail.onmicrosoft.com

RTAuser3@arrabiagmail.onmicrosoft.com

RTAuser4@arrabiagmail.onmicrosoft.com

RTAuser5@arrabiagmail.onmicrosoft.com

RTAuser6@arrabiagmail.onmicrosoft.com

RTAuser7@arrabiagmail.onmicrosoft.com

RTAuser8@arrabiagmail.onmicrosoft.com

RTAuser9@arrabiagmail.onmicrosoft.com

RTAuser10@arrabiagmail.onmicrosoft.com

RTAuser11@arrabiagmail.onmicrosoft.com

RTAuser12@arrabiagmail.onmicrosoft.com

RTAuser13@arrabiagmail.onmicrosoft.com

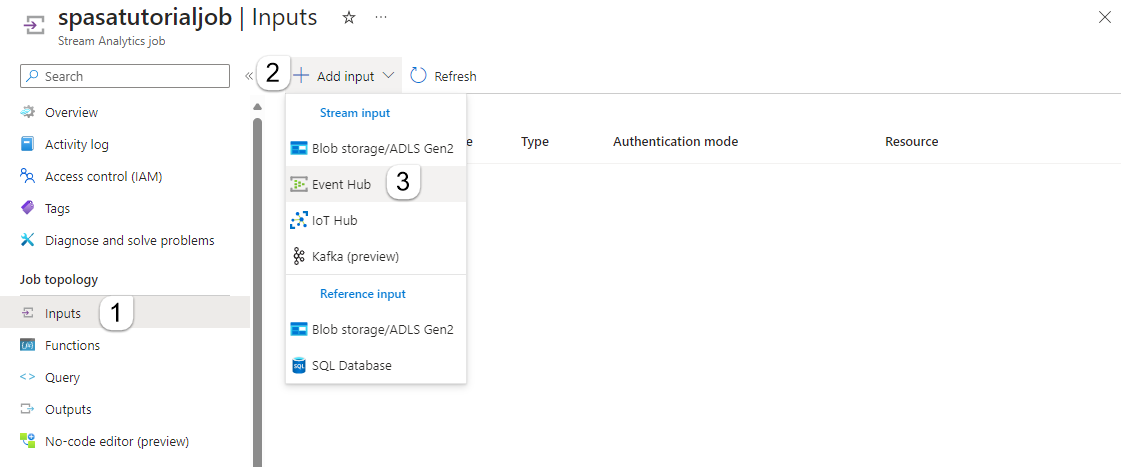
RTAuser14@arrabiagmail.onmicrosoft.com

RTAuser15@arrabiagmail.onmicrosoft.com

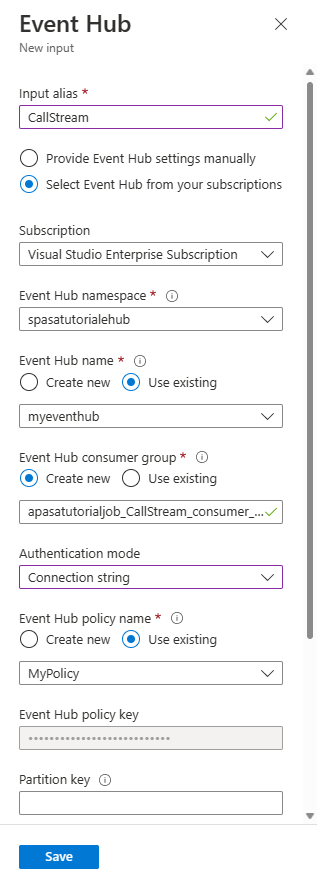
**Configure job input**

The next step is to define an input source for the job to read data using the event hub you created in the previous section.

1. On the **Stream Analytics job** page, in the **Job Topology** section on the left menu, select **Inputs**.
2. On the **Inputs** page, select **+ Add input** and **Event hub**.

[](https://learn.microsoft.com/en-us/azure/stream-analytics/includes/media/event-generator-app/add-input-event-hub-menu.png#lightbox)

1. On the **Event hub** page, follow these steps:
   1. For **Input alias**, enter **CallStream**. Input alias is a friendly name to identify your input. Input alias can contain alphanumeric characters, hyphens, and underscores only and must be 3-63 characters long.
   2. For **Subscription**, select the Azure subscription where you created the event hub. The event hub can be in same or a different subscription as the Stream Analytics job.
   3. For **Event Hubs namespace**, select the Event Hubs namespace you created in the previous section. All the namespaces available in your current subscription are listed in the dropdown.
   4. For **Event hub name**, select the event hub you created in the previous section. All the event hubs available in the selected namespace are listed in the dropdown.
   5. For **Event hub consumer group**, keep the **Create new** option selected so that a new consumer group is created on the event hub. We recommend that you use a distinct consumer group for each Stream Analytics job. If no consumer group is specified, the Stream Analytics job uses the $Default consumer group. When a job contains a self-join or has multiple inputs, some inputs later might be read by more than one reader. This situation affects the number of readers in a single consumer group.
   6. For **Authentication mode**, select **Connection string**. It's easier to test the tutorial with this option.
   7. For **Event hub policy name**, select **Use existing**, and then select the policy you created earlier.
   8. Select **Save** at the bottom of the page.



**Configure job output**

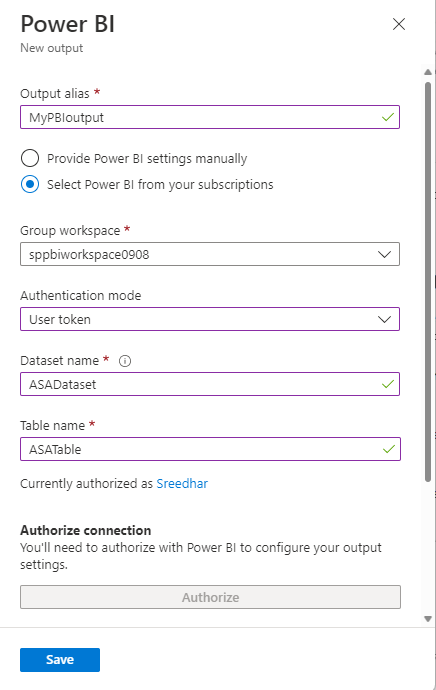
The last step is to define an output sink where the job can write the transformed data. In this tutorial, you output and visualize data with Power BI.

1. From the Azure portal, open **All resources**, and select the *ASATutorial* Stream Analytics job.
2. In the **Job Topology** section of the Stream Analytics job, select the **Outputs** option.
3. Select **+ Add** > **Power BI**.
4. Fill the output form with the following details:

Expand table

| **Setting** | **Suggested value** |
| --- | --- |
| Output alias | MyPBIoutput |
| Group workspace | My workspace |
| Dataset name | ASAdataset |
| Table name | ASATable |
| Authentication mode | User token |

1. Select **Authorize** and follow the prompts to authenticate Power BI.



1. Select **Save** at the bottom of the **Power BI** page.

This tutorial uses the *User token* authentication mode. To use Managed Identity, see [Use Managed Identity to authenticate your Azure Stream Analytics job to Power BI](https://learn.microsoft.com/en-us/azure/stream-analytics/powerbi-output-managed-identity).

**Create queries to transform real-time data**

At this point, you have a Stream Analytics job set up to read an incoming data stream. The next step is to create a query that analyzes the data in real time. The queries use a SQL-like language that has some extensions specific to Stream Analytics.

In this section of the tutorial, you create and test several queries to learn a few ways in which you can transform an input stream for analysis.

The queries you create here will just display the transformed data to the screen. In a later section, you'll write the transformed data to Power BI.

To learn more about the language, see the [Azure Stream Analytics Query Language Reference](https://learn.microsoft.com/en-us/stream-analytics-query/stream-analytics-query-language-reference).

**Test using a pass-through query**

If you want to archive every event, you can use a pass-through query to read all the fields in the payload of the event.

1. Navigate to your Stream Analytics job in the Azure portal and select **Query** under **Job topology** on the left menu.
2. In the query window, enter this query:

SQLCopy

SELECT

\*

FROM

CallStream

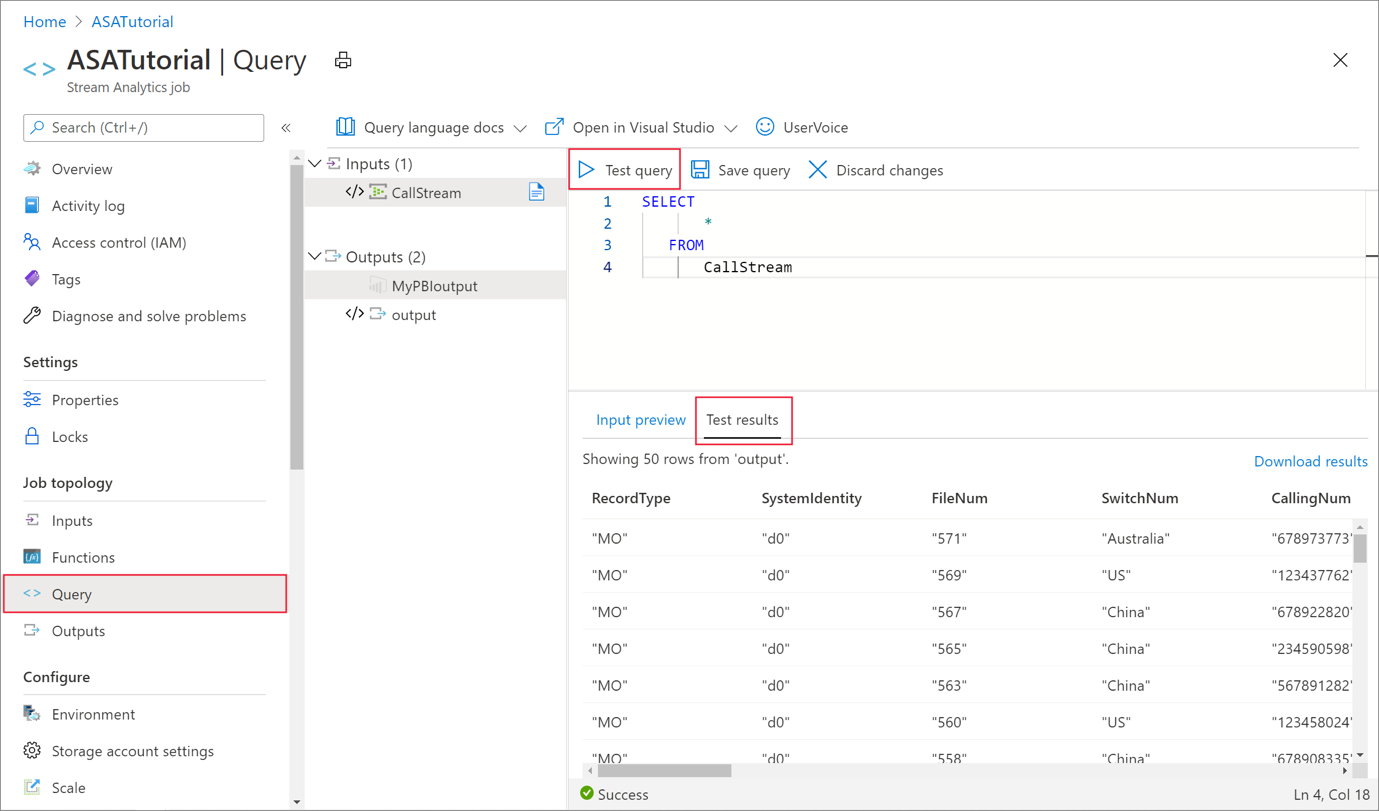
**Note**

As with SQL, keywords are not case-sensitive, and whitespace is not significant.

In this query, CallStream is the alias that you specified when you created the input. If you used a different alias, use that name instead.

1. Select **Test query**.

The Stream Analytics job runs the query against the sample data from the input and displays the output at the bottom of the window. The results indicate that the Event Hubs and the Streaming Analytics job are configured correctly.



The exact number of records you see depends on how many records were captured in the sample.

**Reduce the number of fields using a column projection**

In many cases, your analysis doesn't need all the columns from the input stream. You can use a query to project a smaller set of returned fields than in the pass-through query.

Run the following query and notice the output.

SQLCopy

SELECT CallRecTime, SwitchNum, CallingIMSI, CallingNum, CalledNum

INTO

[MyPBIoutput]

FROM

CallStream

**Count incoming calls by region: Tumbling window with aggregation**

Suppose you want to count the number of incoming calls per region. In streaming data, when you want to perform aggregate functions like counting, you need to segment the stream into temporal units, since the data stream itself is effectively endless. You do this using a Streaming Analytics [window function](https://learn.microsoft.com/en-us/azure/stream-analytics/stream-analytics-window-functions). You can then work with the data inside that window as a unit.

For this transformation, you want a sequence of temporal windows that don't overlap—each window has a discrete set of data that you can group and aggregate. This type of window is referred to as a *Tumbling window*. Within the Tumbling window, you can get a count of the incoming calls grouped by SwitchNum, which represents the country/region where the call originated.

1. Paste the following query in the query editor:

SQLCopy

SELECT

System.Timestamp as WindowEnd, SwitchNum, COUNT(\*) as CallCount

FROM

CallStream TIMESTAMP BY CallRecTime

GROUP BY TUMBLINGWINDOW(s, 5), SwitchNum

This query uses the Timestamp By keyword in the FROM clause to specify which timestamp field in the input stream to use to define the Tumbling window. In this case, the window divides the data into segments by the CallRecTime field in each record. (If no field is specified, the windowing operation uses the time that each event arrives at the event hub. See "Arrival Time vs Application Time" in [Stream Analytics Query Language Reference](https://learn.microsoft.com/en-us/stream-analytics-query/stream-analytics-query-language-reference).

The projection includes System.Timestamp, which returns a timestamp for the end of each window.

To specify that you want to use a Tumbling window, you use the [TUMBLINGWINDOW](https://learn.microsoft.com/en-us/stream-analytics-query/tumbling-window-azure-stream-analytics) function in the GROUP BY clause. In the function, you specify a time unit (anywhere from a microsecond to a day) and a window size (how many units). In this example, the Tumbling window consists of 5-second intervals, so you get a count by country/region for every 5 seconds' worth of calls.

1. Select **Test query**. In the results, notice that the timestamps under **WindowEnd** are in 5-second increments.

**Detect SIM fraud using a self-join**

For this example, consider fraudulent usage to be calls that originate from the same user but in different locations within 5 seconds of one another. For example, the same user can't legitimately make a call from the US and Australia at the same time.

To check for these cases, you can use a self-join of the streaming data to join the stream to itself based on the CallRecTime value. You can then look for call records where the CallingIMSI value (the originating number) is the same, but the SwitchNum value (country/region of origin) isn't the same.

When you use a join with streaming data, the join must provide some limits on how far the matching rows can be separated in time. As noted earlier, the streaming data is effectively endless. The time bounds for the relationship are specified inside the ON clause of the join, using the DATEDIFF function. In this case, the join is based on a 5-second interval of call data.

1. Paste the following query in the query editor:

SQLCopy

SELECT System.Timestamp AS WindowEnd, COUNT(\*) AS FraudulentCalls

INTO "MyPBIoutput"

FROM "CallStream" CS1 TIMESTAMP BY CallRecTime

JOIN "CallStream" CS2 TIMESTAMP BY CallRecTime

ON CS1.CallingIMSI = CS2.CallingIMSI

AND DATEDIFF(ss, CS1, CS2) BETWEEN 1 AND 5

WHERE CS1.SwitchNum != CS2.SwitchNum

GROUP BY TumblingWindow(Duration(second, 1))

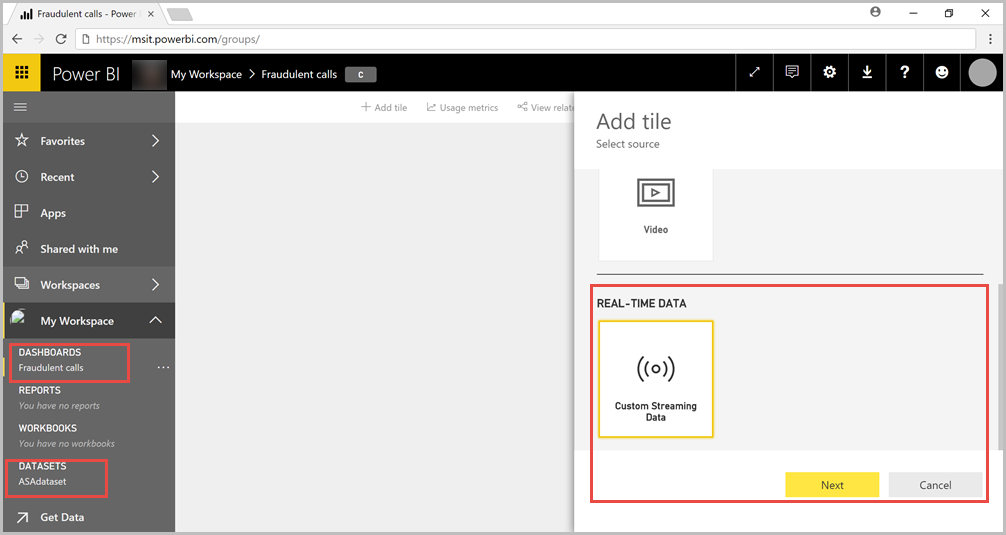
This query is like any SQL join except for the DATEDIFF function in the join. This version of DATEDIFF is specific to Streaming Analytics, and it must appear in the ON...BETWEEN clause. The parameters are a time unit (seconds in this example) and the aliases of the two sources for the join. This function is different from the standard SQL DATEDIFF function.

The WHERE clause includes the condition that flags the fraudulent call: the originating switches aren't the same.

1. Select **Test query**. Review the output, and then select **Save query**.

**Start the job and visualize output**

1. To start the job, navigate to the job **Overview** and select **Start**.
2. Select **Now** for job output start time and select **Start**. You can view the job status in the notification bar.
3. Once the job succeeds, navigate to [Power BI](https://powerbi.com/) and sign in with your work or school account. If the Stream Analytics job query is outputting results, the *ASAdataset* dataset you created exists under the **Datasets** tab.
4. From your Power BI workspace, select **+ Create** to create a new dashboard named *Fraudulent Calls*.
5. At the top of the window, select **Edit** and **Add tile**. Then select **Custom Streaming Data** and **Next**. Choose the **ASAdataset** under **Your Datasets**. Select **Card** from the **Visualization type** dropdown, and add **fraudulent calls** to **Fields**. Select **Next** to enter a name for the tile, and then select **Apply** to create the tile.



1. Follow the step 5 again with the following options:
   * When you get to Visualization Type, select Line chart.
   * Add an axis and select **windowend**.
   * Add a value and select **fraudulent calls**.
   * For **Time window to display**, select the last 10 minutes.
2. Your dashboard should look like the following example once both tiles are added. Notice that, if your event hub sender application and Streaming Analytics application are running, your Power BI dashboard periodically updates as new data arrives.

