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

Learn how to create an end-to-end solution for real-time fraud detection with Azure Stream Analytics. Bring events into an Azure event hub, write Stream Analytics queries for aggregation or alerting, and send the results to an output sink to gain insight over data with real-time processing.

Stream Analytics is a fully managed service providing low-latency, highly available, scalable complex event processing over streaming data in the cloud. For more information, see [Introduction to Azure Stream Analytics](https://azure.microsoft.com/en-in/documentation/articles/stream-analytics-introduction/).

Scenario: Telecommunications and SIM fraud detection in real-time

A telecommunications company has a large volume of data for incoming calls. The company needs the following from its data: \* Pare this data down to a manageable amount and obtain insights about customer usage over time and geographical regions. \* Detect SIM fraud (multiple calls coming from the same identity around the same time but in geographically different locations) in real-time so that they can easily respond by notifying customers or shutting down service.

In canonical Internet of Things (IoT) scenarios there is a ton of telemetry or sensor data being generated – and customers want to aggregate them or alert over anomalies in real-time.

Prerequisites

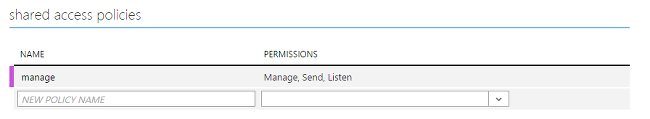
This scenario leverages an event generator located on GitHub. Download it [here](https://github.com/Azure/azure-stream-analytics/tree/master/DataGenerators/TelcoGenerator) and follow the steps in this tutorial to set up your solution.

Create an Azure Event Hubs input and Consumer Group

The sample application will generate events and push them to an Event Hub instance for real-time processing. Service Bus Event Hubs are the preferred method of event ingestion for Stream Analytics and you can learn more about Event Hubs in [Azure Service Bus documentation](https://azure.microsoft.com/documentation/services/service-bus/).

To create an Event Hub:

1. In the [Azure portal](https://manage.windowsazure.com/) click **New** > **App Services** > **Service Bus** > **Event Hub** > **Quick Create**. Provide a name, region, and new or existing namespace to create a new Event Hub.
2. As a best practice, each Stream Analytics job should read from a single Event Hub Consumer Group. We will walk you through the process of creating a Consumer Group below, and you can [learn more about Consumer Groups](https://msdn.microsoft.com/library/azure/dn836025.aspx). To create a Consumer Group, navigate to the newly created Event Hub and click the **Consumer Groups** tab, then click **Create** on the bottom of the page and provide a name for your Consumer Group.
3. To grant access to the Event Hub, we will need to create a shared access policy. Click the **Configure** tab of your Event Hub.
4. Under **Shared Access Policies**, create a new policy with **Manage** permissions.



1. Click **Save** at the bottom of the page.
2. Navigate to the **Dashboard** and click **Connection Information** at the bottom of the page, and then copy and save the connection information.

Configure and start event generator application

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

1. Download the TelcoGenerator solution from <https://github.com/Azure/azure-stream-analytics/tree/master/DataGenerators/TelcoGenerator>.
2. Replace the Microsoft.ServiceBus.ConnectionString and EventHubName values in App.Config with your Event Hub connection string and name.
3. Build the solution to trigger the download of required nuget packages.
4. Start the application. The usage is as follows:

Copy to clipboardCopy

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

The following example will generate 1000 events with a 20 percent probability of fraud over the course of 2 hours.

Copy to clipboardCopy

TelcoDataGen.exe 1000 .2 2

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

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

Create Stream Analytics job

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

Provision a Stream Analytics job

1. In the Azure portal, click **New > Data Services > Stream Analytics > Quick Create**.
2. Specify the following values, and then click **Create Stream Analytics Job**:
   * **Job Name**: Enter a job name.
   * **Region**: Select the region where you want to run the job. Consider placing the job and the event hub in the same region to ensure better performance and to ensure that you will not be paying to transfer data between regions.
   * **Storage Account**: Choose the Azure storage account that you would like to use to store monitoring data for all Stream Analytics jobs running within this region. You have the option to choose an existing storage account or to create a new one.
3. Click **Stream Analytics** in the left pane to list the Stream Analytics jobs.

Stream Analytics service icon

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

Specify job input

1. In your Stream Analytics job click **Inputs** from the top of the page, and then click **Add Input**. The dialog box that opens will walk you through a number of steps to set up your input.
2. Select **Data Stream**, and then click the right button.
3. Select **Event Hub**, and then click the right button.
4. Type or select the following values on the third page:
   * **Input Alias**: Enter a friendly name for this job input such as *CallStream*. Note that you will be using this name in the query later.
   * **Event Hub**: If the Event Hub you created is in the same subscription as the Stream Analytics job, select the namespace that the event hub is in.

If your event hub is in a different subscription, select **Use Event Hub from Another Subscription** and manually enter information for **Service Bus Namespace**, **Event Hub Name**, **Event Hub Policy Name**, **Event Hub Policy Key**, and **Event Hub Partition Count**. \* **Event Hub Name**: Select the name of the Event Hub. \* **Event Hub Policy Name**: Select the event-hub policy created earlier in this tutorial. \* **Event Hub Consumer Group**: Type the Consumer Group created earlier in this tutorial.

1. Click the right button.
2. Specify the following values:
   * **Event Serializer Format**: JSON
   * **Encoding**: UTF8
3. Click the check button to add this source and to verify that Stream Analytics can successfully connect to the event hub.

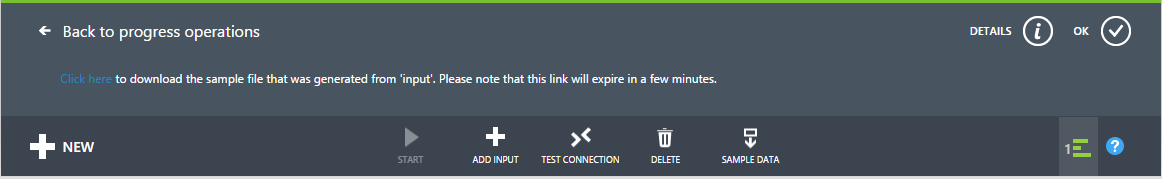
Specify job query

Stream Analytics supports a simple, declarative query model for describing transformations for real-time processing. To learn more about the language, see the [Azure Stream Analytics Query Language Reference](https://msdn.microsoft.com/library/dn834998.aspx). This tutorial will help you author and test several queries over your real-time stream of call data.

Optional: Sample input data

To validate your query against actual job data, you can use the **Sample Data** feature to extract events from your stream and create a .JSON file of the events for testing. The following steps show how to do this and we have also provided a sample [Telco.json](https://github.com/Azure/azure-stream-analytics/blob/master/Sample%20Data/telco.json) file for testing purposes.

1. Select your Event Hub input and click **Sample Data** at the bottom of the page.
2. In the dialog box that appears, specify a **Start Time** to start collecting data from and a **Duration** for how much additional data to consume.
3. Click the check button to start sampling data from the input. It can take a minute or two for the data file to be produced. When the process is completed, click **Details** and download and save the .JSON file that is generated.



Passthrough query

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

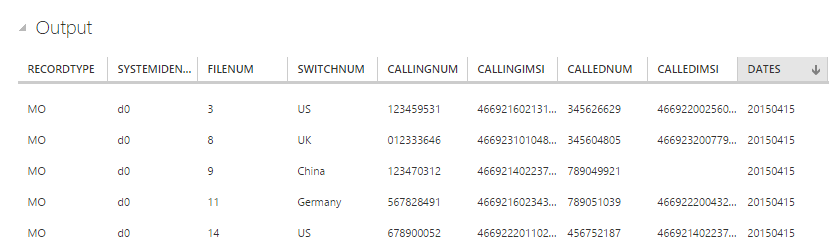
1. Click **Query** from the top of the Stream Analytics job page.
2. Add the following to the code editor:

Copy to clipboardCopy

SELECT \* FROM CallStream

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

1. Click **Test** under the query editor.
2. Supply a test file, either one that you created using the previous steps or use [Telco.json](https://github.com/Azure/azure-stream-analytics/blob/master/Sample%20Data/telco.json).
3. Click the check button and see the results displayed below the query definition.



Column projection

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

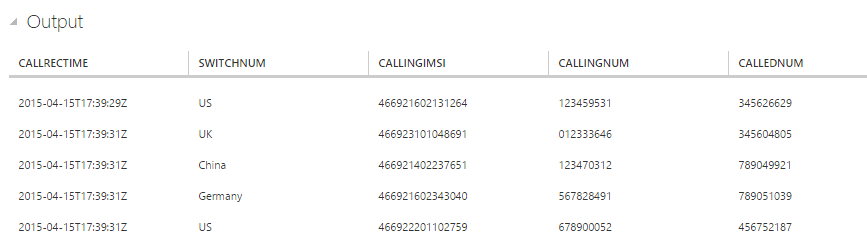
1. Change the query in the code editor to:

Copy to clipboardCopy

SELECT CallRecTime, SwitchNum, CallingIMSI, CallingNum, CalledNum

FROM CallStream

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



Count of incoming calls by region: Tumbling window with aggregation

To compare the amount that incoming calls per region we'll leverage a [TumblingWindow](https://msdn.microsoft.com/library/azure/dn835055.aspx) to get the count of incoming calls grouped by SwitchNum every 5 seconds.

1. Change the query in the code editor to:

Copy to clipboardCopy

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

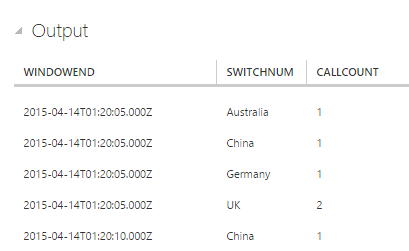
FROM CallStream TIMESTAMP BY CallRecTime

GROUP BY TUMBLINGWINDOW(s, 5), SwitchNum

This query uses the **Timestamp By** keyword to specify a timestamp field in the payload to be used in the temporal computation. If this field wasn't specified, the windowing operation would be performed using the time each event arrived at Event Hub. See ["Arrival Time Vs Application Time" in the Stream Analytics Query Language Reference](https://msdn.microsoft.com/library/azure/dn834998.aspx).

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

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



SIM fraud detection with a Self-Join

To identify potentially fraudulent usage we'll look for calls originating from the same user but in different locations in less than 5 seconds. We [join](https://msdn.microsoft.com/library/azure/dn835026.aspx) the stream of call events with itself to check for these cases.

1. Change the query in the code editor to:

Copy to clipboardCopy

SELECT System.Timestamp as Time, CS1.CallingIMSI, CS1.CallingNum as CallingNum1,

CS2.CallingNum as CallingNum2, CS1.SwitchNum as Switch1, CS2.SwitchNum as Switch2

FROM CallStream CS1 TIMESTAMP BY CallRecTime

JOIN CallStream CS2 TIMESTAMP BY CallRecTime

ON CS1.CallingIMSI = CS2.CallingIMSI

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

WHERE CS1.SwitchNum != CS2.SwitchNum

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



Create output sink

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

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

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

Specify job output

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

Start job for real time processing

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

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

View fraud detection output

Use a tool like [Azure Storage Explorer](https://azurestorageexplorer.codeplex.com/) or [Azure Explorer](http://www.cerebrata.com/products/azure-explorer/introduction) to view fraudulent events as they are written to your output in real-time.

