Anomaly Detection in IoT Telemetry

Azure IoT Workshop

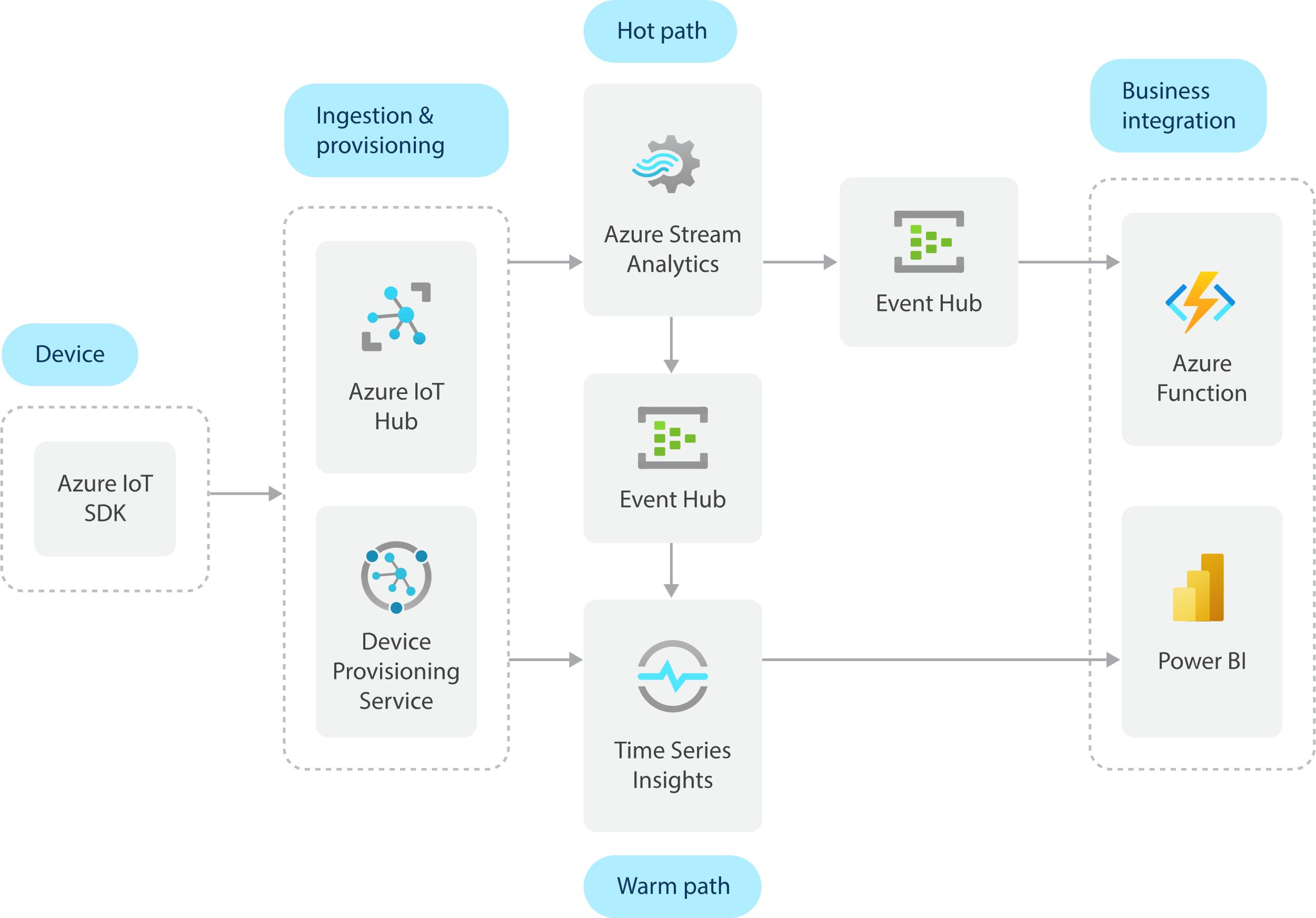
By Jevgenijs Pankovs

# Introduction

The goal of this workshop is to show you in minute details how to implement a typical IoT solution, fulfilling the following requirements:

* An IoT device, sending telemetry data, shall be connected to an Azure IoT Hub
* The telemetry data shall be processed with Azure Stream Analytics service to detect anomalies – spikes and dips or change points
* Visualize telemetry data and anomalies with Azure Time Series Insights
* Visualize telemetry and anomalies with Power BI
* Egress Stream Analytics output to an Azure Function to process anomalies.

The following diagram reflects different Azure components that will be used to architect our IoT solution:



*Figure 1. Diagram of the architecture*

First, we will deploy all the components to the Cloud manually. Then, we will be using a deployment script, creating the same components, to avoid repeating manual operations, to simplify and speed up our work.

Though, we could use a real IoT device (like a Raspberry Pi), we will create a device simulator using Visual Studio Code, Azure IoT SDK and Python or C# languages.

The amount of work is large, that’s why we will split it to a few parts. Here is our plan:

**Part I:**

1. Create a resource group in Azure portal
2. Create an IoT hub
3. Create a Device Provisioning service, and link it to the IoT hub
4. Create an IoT device simulator, connect it to the IoT hub, and send telemetry data
5. Use Azure IoT Explorer to view the telemetry the simulated device is sending

**Part II:**

1. Create a Stream Analytics job, and configure its inputs, outputs, and queries
2. Create a Time Series Insights environment, and configure its event sources
3. Visualize telemetry data in Time Series Insights
4. Visualize data from Time Series Insights in Power BI
5. Create an Azure Function and trigger it in response to Stream Analytics events.

Enough with the introduction, let’s start.

# Hands-on Lab. Part I

## 1. Create a resource group

Let us first create an Azure Resource Group, which will be a logical container for all our components. All the resources in a resource group will have the same life cycle. That means when a resource group is deleted, all resources in the group are deleted as well. The group needs a location to specify where the metadata will be stored, which is necessary for some compliance policies.

1. Open Azure portal on the browser by navigating to <http://portal.azure.com>
2. From the left top side of the portal home page click **+ Create a resource** and select **Resource group** from the features options
3. Select your **Subscription**, enter ***anomaly-detection-rg*** as a **Resource group** name, select ***North Europe*** **Region**, and then select **Review + Create**
4. On the **Summary** page, select **Create**.

Graphical user interface, text, application, email

Description automatically generated

Once the deployment is complete, select **Go to resource** to open new resource group. Click **+ Create** to create a new resource:

Graphical user interface, text, application

Description automatically generated

## 2. Create an IoT hub

Now, we will create an IoT hub, acting as a central message hub for communication between our IoT applications and its attached devices.

Graphical user interface, text, application, email

Description automatically generated

1. Select **IoT Hub** from the list of resources
2. Enter ***anomaly-detection-hub*** as a name of the IoT hub, and then select ***North Europe*** **Region**
3. Press **Next: Networking**, and then select **Next: Management** to continue creating your hub.

### Graphical user interface, text, application, email Description automatically generated

1. Leave **S1: Standard tier** as **Pricing and scale tier**
2. Select **Review + Create** and click **Create**.

Once the IoT hub is created, select **Go to resource** to open the hub.

IoT Hub exposes a built-in endpoint to read the device-to-cloud messages received by your hub. IoT Hub enables you to create consumer groups on the endpoint, enabling multiple consuming applications to read the stream independently.

We are going to create a new consumer group for Azure Stream Analytics service, reading telemetry data from our IoT hub.

1. Go to **Built-in endpoints** tab
2. Enter ***asaconsumergroup*** as a new **Consumer Group**.

## Graphical user interface, text, application Description automatically generated

## 3. Create a Device Provisioning Service

Azure IoT Service offers many ways to connect an IoT device to your IoT hub. For example, you can register your devices in the IoT hub manually and connect them using symmetric keys or X.509 certificates. But if you need to provision thousands (or even millions) of devices, that’s not the case.

We need to realize a scenario when a connected IoT device is automatically registered in our IoT hub. That is why we will use Device Provisioning Service (DPS), supporting IoT Plug and Play architecture. Once a device starts communicating using Plug and Play protocol, it is automatically registered (if it was not) in the IoT hub. That is what we need. But first, let’s create a DPS.

1. In the Azure portal, select **+ Create a resource**, and select **IoT Hub Device Provisioning Service**
2. Select your **Subscription**, select ***anomaly-detection-rg*** as a **Resource group**, and ***North Europe*** as a **Region**
3. Enter ***anomaly-detection-dps*** as a **Name** and then select **Review + Create**
4. On the **Review + Create** page, select **Create**.

Graphical user interface, text, application, email

Description automatically generated

1. Once the DPS instance is created, open it, and go to **Linked IoT hubs** tab
2. Click **+ Add** button to link our previously created IoT hub

Graphical user interface, text, application

Description automatically generated

1. On the **Add link to IoT hub** panel, select your **Subscription**
2. Select ***anomaly-detection-hub*** as a name of the **IoT hub**
3. Select ***iothubowner*** as **Access Policy**, and then click **Save**.

Graphical user interface, text, application, email

Description automatically generated

We’ve just linked our IoT hub to the DPS. If you do not see your hub in the list of linked hubs, click **Refresh**.

Now, we need to create a DPS Enrollment Group. This enrollment group will have a set of credentials for all IoT devices in the same group.

Select **Manage enrollments** tab and click **+ Add enrollment group**.

Graphical user interface, text, application

Description automatically generated

1. On **Add Enrollment Group**, enter ***anomaly-detection-group*** as a **Group name**
2. Select ***Symmetric Key*** as **Attestation Type**
3. Check **Auto-generate keys** checkbox and click **Save**.

Graphical user interface, text, application, email

Description automatically generated

Once you saved your enrollment group, open the enrollment, copy, and save the value of your generated **Primary Key** for future usage. This key is your master group key – we will need it in our IoT device simulator.

Graphical user interface, application, Teams

Description automatically generated

Another important parameter, needed for our device simulator is DPS ID Scope. Go to **Overview** tab and find **ID Scope** value:

Graphical user interface, application

Description automatically generated

Copy and save it somewhere in your notes as **DPS ID Scope** value.

## 4. Create an IoT device simulator

In this section we will create an IoT device simulator sending telemetry data (temperature and humidity) to the IoT hub.

First, you can use Git to clone the repository with the source code. Change the current working directory to the location where you want the cloned directory. Then, type:

**> git clone https://github.com/jevgenij-p/iot-workshop-anomaly-detection.git­**

Downloaded repository contains *source* folder with two sub-folders:

*source\simulator\python  
source\simulator\csharp*

You can choose the version of the IoT device simulator: either written in Python, or C#. We will discuss both.

### 4.1 IoT device simulator. Python version

This is a small application, connecting to an IoT hub, registering a device, and sending some telemetry to the hub. The telemetry contains temperature and humidity values, changing in a narrow range. We can increase or decrease temperature values pressing “+” or “-“ keys and ENTER key, or to quit the application pressing the “q” key.

But before running the simulator ensure that your system meets all prerequisites.

#### 4.1.1 Prerequisites

Install the latest version of [Python](https://www.python.org/downloads/) and add it to the Windows *Path* variable.

Change the current working directory to the *source\simulator\python* directory, and create a virtual environment:

**> python -m venv env**

Activate the virtual environment by the following command:

**> .\env\Scripts\activate**

Install required packages:

**(env) > pip3 install azure-iot-device  
(env) > pip3 install asyncio**

#### 4.1.2 Run IoT device simulator

Before you run the application, you shall set some system variables. Please find the [Primary Key](#Primary_Key) and [DPS ID Scope](#DPS_ID_Scope) you saved before. You need to replace **[Primary\_Key]** and **[DPS\_ID\_Scope]** with the saved values in the following commands:

**(env) > SET DPS\_DEVICE\_ID=device001**

**(env) > SET DPS\_ID\_SCOPE=[DPS\_ID\_SCOPE]**

**(env) > SET DPS\_PRIMARY\_KEY=[PRIMARY\_KEY]**

For example:

**(env) > SET DPS\_DEVICE\_ID=device001  
(env) > SET DPS\_ID\_SCOPE=0ne004269B3**

**(env) > SET DPS\_PRIMARY\_KEY=cwB2AGMAXgBbAC8ASQA8AG8AJQBsAHwAeQBoACQAKQA3ADAAOABlADsAZQBpACsAQwBpAFAAdABfAEkAVQByAA==**

Run the simulator:

**(env) > python telemetry.py**

If everything is set up correctly, you will see messages sent to the IoT hub every two seconds:

Text

Description automatically generated

If you are not familiar with Python, you can use C# version of the simulator.

### 4.2 IoT device simulator. C# version

The source code of the C# simulator is in the *source\simulator\csharp\telemetry* directory. The logic of this application is absolutely the same as of its Python version: it connects to an IoT hub, registers a device, and sends telemetry to the hub. We can increase or decrease temperature values pressing “+” or “-“ keys and ENTER key, or to quit the application pressing *Ctrl+C*.

But before running the simulator check out the following prerequisites.

#### 4.2.1 Prerequisites

Install [Visual Studio Code](https://code.visualstudio.com/) first and then add the following extensions:

* [C# for Visual Studio Code](https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp)
* [Azure IoT Tools for Visual Studio Code](https://marketplace.visualstudio.com/items?itemName=vsciot-vscode.azure-iot-tools) (optional)

The IoT device simulator is a .NET 5.0 console application. So, the .NET Runtime 5.0 shall be installed on your workstation.

Change the current working directory to the *source\simulator\csharp\telemetry* directory, and restore the dependencies of the project:

**> dotnet restore**

#### 4.2.2 Run IoT device simulator

Before you run the application, you shall set some system variables. Please find the [Primary Key](#Primary_Key) and [DPS ID Scope](#DPS_ID_Scope) you saved before. You need to replace **[Primary\_Key]** and **[DPS\_ID\_Scope]** with the saved values in the following commands:

**> SET DPS\_DEVICE\_ID=device001**

**> SET DPS\_ID\_SCOPE=[DPS\_ID\_SCOPE]**

**> SET DPS\_PRIMARY\_KEY=[PRIMARY\_KEY]**

For example:

**> SET DPS\_DEVICE\_ID=device001  
> SET DPS\_ID\_SCOPE=0ne004269B3**

**> SET DPS\_PRIMARY\_KEY=cwB2AGMAXgBbAC8ASQA8AG8AJQBsAHwAeQBoACQAKQA3ADAAOABlADsAZQBpACsAQwBpAFAAdABfAEkAVQByAA==**

Run the simulator:

**> dotnet run**

The simulator will be sending messages to the IoT hub every two seconds:

Text

Description automatically generated

Press “+” or “-“ and ENTER to increase or decrease temperature. Press *Ctrl+C* to quit the application.

## 5. Use Azure IoT Explorer to view the telemetry

It looks our simulator is sending some data to the IoT hub. But it would be great to see what exactly it is sending. Fortunately, there’s a tool to view the telemetry the devices are sending: [Azure IoT Explorer](https://github.com/Azure/azure-iot-explorer). Download and install its [latest version](https://github.com/Azure/azure-iot-explorer/releases).

To establish connection with your IoT hub, you will need to get an IoT hub connection string:

1. In the Azure portal, select **anomaly-detection-hub**
2. Select **Shared access policies** tab
3. Click on **iothubowner** policy name to open its properties
4. Copy **Primary connection string** and save it in your notes

Graphical user interface, text, application, email

Description automatically generated

Open the Azure IoT Explorer, select **IoT hubs** tab, add the connection string you have just copied, and click **Save**.

After the Azure IoT Explorer connects to your IoT hub, it displays the devices registered with your IoT hub. Select **device001** in the list and click **Telemetry** tab on the left:

Graphical user interface, text, application, email

Description automatically generated

Ensure your IoT device simulator is running and click **Start** to browse received events. If **Show system properties** checkbox is unchecked, you will see received messages like this:

**{**

**"body": {**

**"temperature": 20.08,**

**"humidity": 59.59**

**},**

**"enqueuedTime": "Wed Dec 01 2021 13:38:51 GMT+0200 (Eastern European Standard Time)",**

**"properties": {}**

**}**

Check **Show system propertie**s checkbox to see all message properties:

**{**

**"body": {**

**"temperature": 19.23,**

**"humidity": 60.59**

**},**

**"enqueuedTime": "Wed Dec 01 2021 13:38:35 GMT+0200 (Eastern European Standard Time)",**

**"properties": {},**

**"systemProperties": {**

**"iothub-connection-device-id": "device001",**

**"iothub-connection-auth-method": "{\"scope\":\"device\",\"type\":\"sas\",\"issuer\":\"iothub\",\"acceptingIpFilterRule\":null}",**

**"iothub-connection-auth-generation-id": "637739461643697540",**

**"iothub-enqueuedtime": 1638358715950,**

**"iothub-message-source": "Telemetry",**

**"contentType": "application/json",**

**"contentEncoding": "utf-8"**

**}**

**}**

I want to point to one important message property:

**"iothub-connection-device-id": "device001"**

This is a *device id* property, which will be used later in Azure Stream Analytics queries.

Now you know how to create an IoT hub, connect IoT devices to it using Device Provisioning service, send telemetry, and browse sent messages in the Azure IoT Explorer. Further we will talk about how to visualize telemetry data.

# Hands-on Lab. Part II

## 6. Create a Stream Analytics job

We are going to receive telemetry stream (temperature and humidity) and detect in real time sudden temperature spikes or dips, using Stream Analytics Query Language.

Azure Stream Analytics offers built-in machine learning based **AnomalyDetection\_SpikeAndDip** function (see [Anomaly detection in Azure Stream Analytics](https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-machine-learning-anomaly-detection)) which returns two columns:

**IsAnomaly** - a BIGINT (0 or 1) indicating if the event was anomalous or not, and  
**Score** - the computed p-value score (float) indicating how anomalous an event is.

We will output the results into an Event Hub so the events corresponding to anomalies can be consumed by other components: Time Series Insights, Power BI, and Azure Functions.

Now, we will set up a Stream Analytics job with this anomaly detection function to read telemetry from our IoT Hub and detect anomalies.

1. In the Azure portal, select **+ Create a resource**, and select **Stream Analytics job**
2. Enter ***streamjob*** as a **Job name**
3. Select your **Subscription**, select ***anomaly-detection-rg*** as a **Resource group**, and ***North Europe*** as a **Region**
4. Click **Create**.

Graphical user interface, text, application

Description automatically generated

Once our Stream Analytics job is created, we need to configure it – create an **Input**, **Output**, and a **Query**.

### 4.1 Configure Stream Analytics Input

Our next step is to define an input source for the job to read data from the IoT hub.

1. Select **Inputs** tab
2. Click **+ Add stream input** and select **IoT Hub.** IoT Hub panel will be opened (see figure below)
3. Enter ***input*** as **Input alias**
4. Select your **Subscription**
5. Select ***anomaly-detection-hub*** as an **IoT Hub**
6. Select ***asaconsumergroup*** as a **Consumer group**
7. Select ***iothubowner*** as a **Shared access policy name**
8. Click **Save**

Graphical user interface, application

Description automatically generated

We want to detect anomalies in temperature and feed these anomalous “events” into other systems or services: Time Series Insights and Azure Functions. Stream Analytics supports many types of outputs: Azure Data Lake, Azure SQL Database, Azure Synapse Analytics, Blob and Table storages, Event Hub, Service Bus, Power BI, Cosmos DB, and Azure Functions. We will use two Event Hubs as outputs for our job. One – for Time Series Insights, and second – for an Azure Function.

### 4.2 Configure Stream Analytics Outputs

First, we need to create a new Event Hub namespace in which we create our event hubs.

1. Go to **anomaly-detection-rg** resource group, click **+ Create**, and select **Event Hubs**
2. Select your **Subscription**, select ***anomaly-detection-rg*** as a **Resource group**, and select ***North Europe*** as **Location**
3. Enter a unique name for the namespace, and then select **Review + Create**
4. On the **Review + Create** page, select **Create**.

Graphical user interface, text, application, email

Description automatically generated

Once the deployment is complete, select **Go to resource** to open the Event Hubs Namespace. We will create two event hubs: ***eventhub1*** as a data source for Time Series Insights, and ***eventhub2*** as a data source for Azure Functions.

1. Select **Event Hubs** tab and click **+ Event Hub**
2. Enter ***eventhub1*** as a Name and click **Create.**

Graphical user interface, text, application, email

Description automatically generated

1. Select **Event Hubs** tab and click **+ Event Hub**
2. Enter ***eventhub2*** as a Name and click **Create**.

Graphical user interface, text, application, email

Description automatically generated

To provide access to the Time Series Insights, we will need to create a Shared Access Policy.

1. Select **Shared access policies** tab and click **+ Add**
2. Enter ***eventHubSharedAccessPolicy*** as a **Policy name**
3. Check **Manage** checkbox and click **Create**.

Graphical user interface, text, application, chat or text message

Description automatically generated

Now we have two event hubs, which will be used as Stream Analytics job outputs.

1. Select **Outputs** tab, click **+ Add**, and select **Event Hub**
2. Enter ***output1*** as an **Output alias**
3. Select your **Subscription**
4. Select the name of the **Event Hub namespace** your created
5. Select ***eventhub1*** as an **Event Hub name**
6. Select ***Connection string*** as an **Authentication mode**
7. Select ***eventHubSharedAccessPolicy*** as an **Event Hub policy name** and click **Save**.

Graphical user interface, application

Description automatically generated

Repeat the same steps for the second output:

1. Click **+ Add**, and select **Event Hub**
2. Enter ***output2*** as an **Output alias**
3. Select your **Subscription**
4. Select the name of the **Event Hub namespace** your created
5. Select ***eventhub2*** as an **Event Hub name**
6. Select ***Connection string*** as an **Authentication mode**

Select ***eventHubSharedAccessPolicy*** as an **Event Hub policy name** and click **Save**.

Graphical user interface, application

Description automatically generated

### 4.3 Create Stream Analytics query

We will be using a Stream Analytics job to send telemetry data to two different outputs using a SQL-like query language (check out [*Stream Analytics Query Language Reference*](https://docs.microsoft.com/en-us/stream-analytics-query/stream-analytics-query-language-reference) to learn more about that).

If we would need to send data to multiple output targets without any transformations, we could use a query like:

SELECT

\*

INTO output1

FROM input;

SELECT

\*

INTO output2

FROM input;

Where **input** is the **Input alias,** we created on the step [*Configure Stream Analytics Input*](#_4.1_Configure_Stream), which is our IoT hub, in fact. **output1** and **output2** are **Output aliases**, pointing to **eventhub1** and **eventhub2**. If we run this query, telemetry stream will be sent to these event hubs.

IoT hub telemetry stream contains many properties (see [*Configure an IoT Hub as a data stream input*](https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-define-inputs#configure-an-iot-hub-as-a-data-stream-input)). If we need to filter only some of them, we could create a query like this:

But our goal is to detect anomalies in temperature. The solution for this use-case is [*AnomalyDetection\_SpikeAndDip*](https://docs.microsoft.com/en-us/stream-analytics-query/anomalydetection-spikeanddip-azure-stream-analytics) function. We can write a query:

SELECT

IoTHub.ConnectionDeviceId AS id,

EventEnqueuedUtcTime AS time,

CAST(temperature AS float) AS temperature,

**AnomalyDetection\_SpikeAndDip(CAST(temperature AS float), 95, 120, 'spikesanddips')**

**OVER(PARTITION BY id LIMIT DURATION(second, 1200)) AS SpikeAndDipScores**

INTO output1

FROM input;

SELECT

IoTHub.ConnectionDeviceId AS id,

EventEnqueuedUtcTime AS time,

CAST(temperature AS float) AS temperature

INTO output1

FROM input;

Where we pass four parameters to the **AnomalyDetection\_SpikeAndDip** function:

|  |  |
| --- | --- |
| **CAST(temperature AS float)** | the event column, containing temperature |
| **95** | confidence - a percentage number from 1.00 to 100. The lower the confidence, the higher the number of anomalies detected, and vice versa |
| **120** | historySize - the number of events in a sliding window that the model continuously learns from and uses for scoring the next event for anomalousness |
| **'spikesanddips'** | mode - a string parameter whose value is "spikes", "dips", or "spikesanddips" |

Maybe we will need to modify these parameters later, but let’s leave it so far.

The following expression

**OVER(PARTITION BY id LIMIT DURATION(second, 600)) AS SpikeAndDipScores**

specifies the size of the sliding window within Stream Analytics in terms of time. The recommended size of this time window is the equivalent of the time it takes to generate *historySize* number of events in steady state.

The **AnomalyDetection\_SpikeAndDip** function returns a record containing the following columns:

|  |  |
| --- | --- |
| **IsAnomaly** | A BIGINT (0 or 1) indicating if the event was anomalous or not |
| **Score** | The computed p-value score indicating how anomalous an event is. Lower scores mean a higher probability that the event is anomalous |

We will use the following query:

WITH AnomalyDetectionStep AS

(

SELECT

IoTHub.ConnectionDeviceId AS id,

EventEnqueuedUtcTime AS time,

CAST(temperature AS float) AS temp,

AnomalyDetection\_SpikeAndDip(CAST(temperature AS float), 95, 120, 'spikesanddips')

OVER(PARTITION BY id LIMIT DURATION(second, 600)) AS SpikeAndDipScores

FROM input

WHERE temperature IS NOT NULL

)

SELECT

id AS 'device-id',

time,

CAST(GetRecordPropertyValue(SpikeAndDipScores, 'IsAnomaly') AS bigint) AS temperature\_anomaly,

CAST(GetRecordPropertyValue(SpikeAndDipScores, 'Score') AS float) AS temperature\_anomaly\_score

INTO output1

FROM AnomalyDetectionStep;

SELECT

id AS 'device-id',

time,

temp AS temperature,

CAST(GetRecordPropertyValue(SpikeAndDipScores, 'IsAnomaly') AS bigint) AS temperature\_anomaly,

CAST(GetRecordPropertyValue(SpikeAndDipScores, 'Score') AS float) AS temperature\_anomaly\_score,

ISFIRST(mi, 1) OVER(PARTITION BY id WHEN CAST(GetRecordPropertyValue(SpikeAndDipScores, 'IsAnomaly') AS bigint) = 1) AS first

INTO output2

FROM AnomalyDetectionStep

WHERE CAST(GetRecordPropertyValue(SpikeAndDipScores, 'IsAnomaly') AS bigint) = 1

*Figure 2. Stream Analytics job query*

which will send events containing the following columns to event hubs:

|  |  |
| --- | --- |
| **device-id** | The device ID |
| **time** | The date and time that the event was received by the IoT Hub |
| **temperature** | The temperature |
| **temperature\_anomaly** | An integer, 0 or 1, indicating whether the event is anomalous or not |
| **temperature\_anomaly\_score** | A score indicating how anomalous the event is |
| **first** | An integer. 1 if the event is the first event within a given fixed interval, or 0 otherwise |

Now we are ready to configure a query in our Stream Analytics job.

1. Select **Query** tab of the job
2. Paste the query we prepared (*Figure 2*) to the query window and click **Save query**

Graphical user interface, application

Description automatically generated

We just configured the Stream Analytics job. The job is not running yet – we will start it later, when we create our IoT device simulator. But now we are going to configure Time Series Insights.

## 5. Create Time Series Insights

We can use Azure Time Series Insights (TSI) to visualize telemetry data that is generated by IoT devices. It can be easily connected to IoT Hubs or Event Hubs to present how their messages change over time. TSI provides visualization through the Azure Time Series Insights Explorer.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

# Send telemetry data

cd iot-workshop-anomaly-detection\source\send-telemetry  
.\env\Scripts\activate  
SET DPS\_ID\_SCOPE=0ne004269B3  
SET DEVICE\_ID=device001  
SET DPS\_PRIMARY\_KEY=cwB2AGMAXgBbAC8ASQA8AG8AJQBsAHwAeQBoACQAKQA3ADAAOABlADsAZQBpACsAQwBpAFAAdABfAEkAVQByAA==  
python telemetry.py

# References

1. [Download Python](https://www.python.org/downloads/)
2. [Visual Studio Code](https://code.visualstudio.com/)
3. [Overview of Azure IoT Device SDKs](https://docs.microsoft.com/en-us/azure/iot-develop/about-iot-sdks)
4. [Azure IoT Samples for C# (.NET)](https://github.com/Azure-Samples/azure-iot-samples-csharp)
5. [Azure IoT Explorer (preview)](https://github.com/Azure/azure-iot-explorer)
6. [Download Azure IoT Explorer](https://github.com/Azure/azure-iot-explorer/releases)
7. [Quickstart: Set up the IoT Hub Device Provisioning Service with the Azure portal](https://docs.microsoft.com/en-us/azure/iot-dps/quick-setup-auto-provision)
8. [What is Azure IoT Hub Device Provisioning Service?](https://docs.microsoft.com/en-us/azure/iot-dps/about-iot-dps)
9. [How to provision devices using symmetric key enrollment groups](https://docs.microsoft.com/en-us/azure/iot-dps/how-to-legacy-device-symm-key?tabs=windows)
10. [Quickstart: Create a Stream Analytics job by using the Azure portal](https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-quick-create-portal)
11. [Anomaly detection in Azure Stream Analytics](https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-machine-learning-anomaly-detection)
12. [AnomalyDetection\_SpikeAndDip Azure Stream Analytics function](https://docs.microsoft.com/en-us/stream-analytics-query/anomalydetection-spikeanddip-azure-stream-analytics)
13. [Stream Analytics Query Language Reference](https://docs.microsoft.com/en-us/stream-analytics-query/stream-analytics-query-language-reference)
14. [Create an Azure Time Series Insights Gen2 environment using the Azure portal](https://docs.microsoft.com/en-us/azure/time-series-insights/how-to-create-environment-using-portal)