Anomaly Detection in IoT Telemetry

Azure IoT Workshop

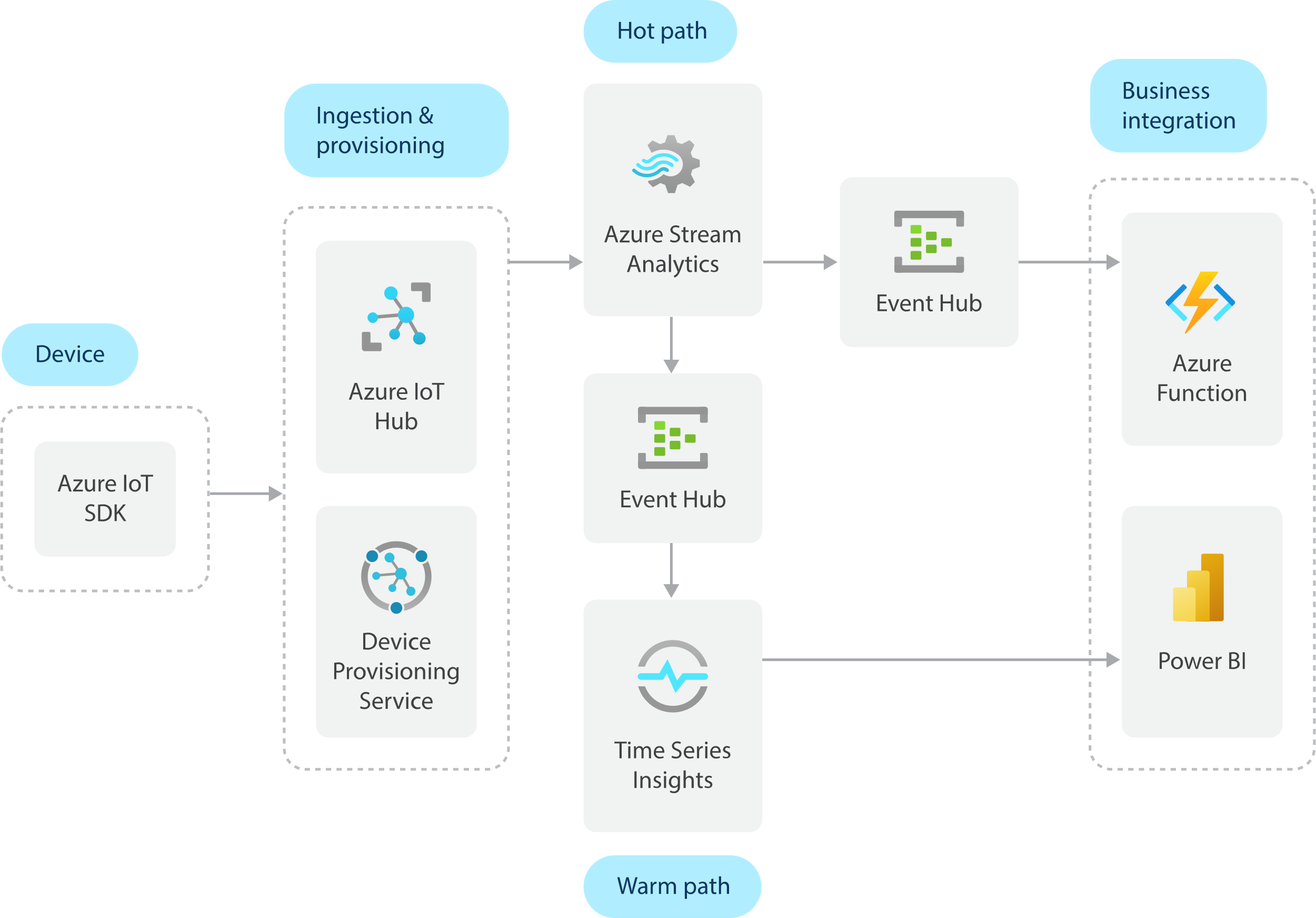
By Jevgenijs Pankovs

# 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

**Part III:**

1. Create a script, deploying Azure resources

**Part IV:**

1. 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.

Open a Command Prompt window. 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"**

**}**

**}**

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**.

### 6.1 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.

### 6.2 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 name**, 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

### 6.3 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:

SELECT

IoTHub.ConnectionDeviceId AS id,

EventEnqueuedUtcTime AS time,

CAST(temperature AS float) AS temperature

INTO output1

FROM input;

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, 240)) AS SpikeAndDipScores**

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, 240)) 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 (120 events \* 2 sec/per event = 240 seconds).

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 temperature,

CAST(humidity AS float) AS humidity,

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

OVER(PARTITION BY id LIMIT DURATION(second, 240)) AS SpikeAndDipScores

FROM input

WHERE temperature IS NOT NULL

)

SELECT

id AS 'device-id',

time,

temperature,

humidity,

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,

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** | Device ID |
| **time** | The date and time that the event was received by the IoT Hub |
| **temperature** | Temperature |
| **humidity** | Humidity |
| **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 the 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

To ensure that everything is configured correctly, we can test our query. Start your IoT device simulator and click **Test query**. If you do not see test results – do not worry – it can take a few minutes to receive events from input.

Table

Description automatically generated

We just configured the Stream Analytics job. The job is not running yet – we need to start it. Select **Overview** tab and click **Start**. The Stream Analytics job will start processing telemetry data and writing query results to the Event Hubs we created early.

Graphical user interface, application

Description automatically generated

## 7. 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. But first, we need to create a TSI service.

1. In the Azure portal, select **+ Create a resource**, and select **Time Service Insights**
2. Select your **Subscription**, select ***anomaly-detection-rg*** as a **Resource group**, and ***North Europe*** as a **Region**
3. Enter ***time-series-insights*** as an **Environment name**
4. Enter ***device-id*** as a **Property name** (we specified it in our Stream Analytics job query)
5. Graphical user interface, text, application, email

   Description automatically generatedEnter a unique **Storage account name**, and then select **Next: Event Source**
6. Enter ***ioteventsource*** as a **Name**
7. Select the **Event Hub namespace** you created as a container for Event hubs in the [Configure Stream Analytics outputs](#_6.2_Configure_Stream) section
8. Select ***eventhub1*** as an **Event Hub name**
9. Select ***eventHubSharedAccessPolicy*** as an **Event Hub access policy name**
10. Select ***$Default*** as an **Event Hub consumer group**, and then select **Review + Create**
11. On the **Summary** page, select **Create**.

Graphical user interface, text, application

Description automatically generated

Once the TSI is created, open it.

Graphical user interface, text, application, email

Description automatically generated

## 8. Visualize telemetry data in Time Series Insights

The TSI Explorer is a graphical tool visualizing telemetry using different chart types. We can define a hierarchy for the telemetry data ingested. We can contextualize the telemetry applying a Time Series Model to simplify finding and analyzing Time Series data.

Check that your IoT device simulator is still running, select **Overview** tab in Time Series Insights and click **Go to TSI Explorer**.

The TSI Explorer shall display only one Time Series instance for **device001** – our simulated IoT device ID:

Graphical user interface, application, table

Description automatically generated

1. Select **device001** – a pop up menu with available streams will be displayed
2. Add **humidity**, **temperature**, **temperature\_anomaly** and **temperature\_anomaly\_score**.

All four streams shall be displayed. But since the temperature is varying in a narrow range, the **temperature\_anomaly** will be zero.

Graphical user interface, text, application, email

Description automatically generated

We can change this situation. Switch to the console application, running your IoT device simulator and try to increase or decrease temperature with “+” and “-“ keys:

A screenshot of a computer

Description automatically generated with medium confidence

You should see (with some time lag) how **temperature\_anomaly** and **temperature\_anomaly\_score** is changing over time.

Graphical user interface, application

Description automatically generated

The **temperature\_anomaly** = 1 indicates the temperature is anomalous. Zero level means normal temperature.

## 9. Visualize data from Time Series Insights in Power BI

Azure Time Series Insights is an analytics, storage, and visualization service that allows to explore billions of IoT events simultaneously. Power BI is more familiar tool for business needs, for users working with IoT data to create reports and dashboards.

The TSI Explorer provides a **Power BI Connector**, which supports export both your recent and historical data out to Power BI. Let me illustrate it on a small example.

1. Open the TSI Explorer, displaying your telemetry data
2. Navigate to the **More actions** dropdown menu and select **Connect to Power BI**

Graphical user interface, application

Description automatically generated

The TSI Explorer generates a **Custom query** for the Power BI Connector to use. See more about how to use the Power BI Connector in the article [Visualize data from Azure Time Series Insights in Power BI](https://docs.microsoft.com/en-us/azure/time-series-insights/how-to-connect-power-bi).

1. Choose a **Store type**: **Warm Store** for the latest data or **Cold Store** for historical data.
2. Click **Copy query to clipboard**
3. Run your Power BI Desktop application and select **Home** tab
4. Select **Get Data**, then select **More**
5. Search for *Time Series*, select **Azure Time Series Insights (Beta)**, and click **Connect**

Graphical user interface, text, application

Description automatically generated

1. Paste the query you copied from the TSI Explorer into the **Custom Query** field, then click **OK**
2. Press **Load** to load data into Power BI

When you have imported the data into Power BI, you can create a report, for example, like this one:

Chart

Description automatically generated

Once you have created your report, you can save it or publish it to Power BI Reporting Services to share it with others.

# Hands-on Lab. Part III

## 10. Create a script, deploying Azure resources

As you already noticed, manual creating of Azure resources is boring and takes too much time. To automate this process, we can use [Azure CLI commands](https://docs.microsoft.com/en-us/cli/azure/iot?view=azure-cli-latest) for Azure IoT. We can create and configure our IoT Hub, Stream Analytics and Time Series Insights using a single PowerShell script, deploying all the resources to the cloud. But before we run it, we need to check some prerequisites.

### 10.1 Prerequisites

We will need the following software:

* The latest version of [PowerShell on Windows](https://docs.microsoft.com/en-us/powershell/scripting/install/installing-powershell-on-windows?view=powershell-7.2)
* [Azure CLI on Windows](https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-windows?tabs=azure-cli)
* [Microsoft Azure IoT extension for Azure CLI](https://github.com/Azure/azure-iot-cli-extension)
* [PowerShell Language Support for Visual Studio Code](https://marketplace.visualstudio.com/items?itemName=ms-vscode.PowerShell)

I recommend using Visual Studio Code for work with PowerShell scripts.

### 10.2 Deployment script

Change the current working directory to the *source\scripts* folder. There are two files:

* *deployment.ps1*
* *functions.ps1*

Open the files in an editor. The *functions.ps1* is an auxiliary script with PowerShell functions, generating random strings. *deployment.ps1* is more interesting for us.

This is a script deploying the following resources to the cloud:

1. Resource group ***anomaly-detection-rg***
2. IoT Hub ***anomaly-detection-hub***
3. IoT Hub device provisioning service ***anomaly-detection-dps***
4. Stream Analytics job ***streamjob***
5. Time Series Insights Gen2 ***time-series-insights***

Before you run the script, open Azure portal, and make sure that the ***anomaly-detection-rg*** resource group does not exist. If it exists – delete it.

Open a Command Prompt window and run the command:

**> pwsh .\deployment.ps1**

The Azure CLI will open your default browser and load an Azure sign-in page. Sign in with your Azure account credentials in the browser.

Text

Description automatically generated

After successful sign-in, the script will create required resources in the ***anomaly-detection-rg*** resource group. That can take a few minutes (around eight minutes in my case). You can take a cup of coffee while robots are working 😊.

Text

Description automatically generated

In the end, the script will print out the **DPS ID Scope** and **DPS Primary Key**, which is used in our IoT device simulator (see section [Run IoT device simulator](#_4.2.2_Run_IoT)).

To check that everything is created, open the ***anomaly-detection-rg*** resource group in Azure portal and select **Overview** tab. You shall see resources, created by the script.

Graphical user interface, text, application, email

Description automatically generated

Now, you can run your IoT device simulator, experiment with the Stream Analytics, Time Series Insights or Power BI. When you finish your experiments, I recommend you delete the ***anomaly-detection-rg*** resource group to clean up the Azure resources to avoid extra costs.

# Hands-on Lab. Part IV

## 11. Create an Azure Function

Let us create an Azure Function that would send out email notification when the telemetry value (temperature) sent out by our simulated IoT device is anomalous.

### 11.1 Prerequisites

We will create this function using Visual Studio Code. You should install [Node.JS](https://docs.npmjs.com/downloading-and-installing-node-js-and-npm) which includes *npm*.

Then, run the following command to install the Azure Functions Core Tools package:

**> npm install -g azure-functions-core-tools@3 --unsafe-perm true**

We will need an [Azure Functions extension](https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-azurefunctions) for Visual Studio Code to develop and deploy functions to Azure.

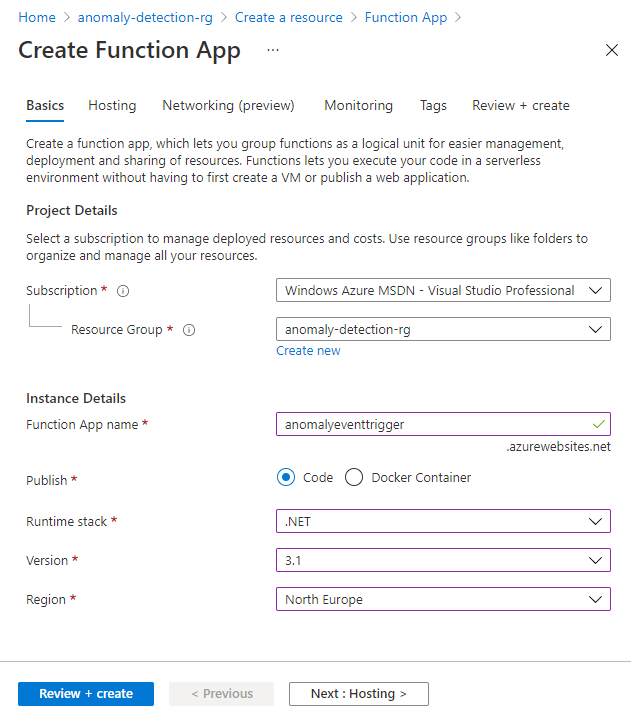
We already have such a function in the *source\functionapp* directory. But we will use this code just as an example and create a new Azure Function in another directory from scratch.

Before we continue, ensure all required Azure resources (IoT hub, DPS, TSI, and so on) exist in the ***anomaly-detection-rg*** resource group in the Azure portal. If not, recreate them manually, or using the PowerShell script (*deployment.ps1*).

### 11.2 Create an Azure Function App in the portal

First, we must have a function app to host the execution of our functions. A function app lets us run our code in a serverless environment.

1. Open Azure portal on the browser
2. Go to **anomaly-detection-rg** resource group, click **+ Create**, and select **Function App**
3. Select your **Subscription**, select ***anomaly-detection-rg*** as a **Resource group**
4. Enter a unique name (like ***anomalyeventtrigger*** in this example) as a **Function App name**
5. Select ***.NET*** as **Runtime stack** and ***3.1*** as **Version**
6. Select ***North Europe*** as **Region**
7. Click **Review + Create**
8. On the **Review + Create** page, select **Create**.



Once the deployment is complete, select **Go to resource** to open new Function App.

### 11.3 Create an Event Hub Trigger

Now we will learn how to create an Azure Event Hub Trigger. The function trigger will respond to an event sent to the **eventhub2** stream. When the function is triggered, the message passed to the function will be sent to an email account.

Open Visual Studio Code and select **Azure** tab on the left navigation menu.

Graphical user interface, text, application

Description automatically generated

Select your Azure subscription. You shall see a function app you just created.

1. Click the *Create New Project…* icon  in the **Functions** panel and select an empty folder (for example, ***myapp***) that will contain your function project
2. Select a language **C#**
3. Select a .NET Runtime **.NET Core 3 LTS**
4. Select a template for your project’s first function **Azure Event Hub trigger**
5. Provide a function name **SendEmailOnEventTrigger**
6. Provide a namespace **IoTLab.Workshop**
7. Select **+ Create new local app setting**
8. Select an event hub namespace
9. Select an event hub **eventhub2**
10. Select an event hub policy **eventHubSharedAccessPolicy**
11. Skip the step, selecting a storage account for internal use.

New local project will be created in the selected folder. If you do not see it in the **Azure: Functions** panel, click *Refresh* icon.

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

Description automatically generated

1. Open created *SendEmailOnEventTrigger.cs* in the Visual Studio Code
2. Find and open *source/functionapp/SendEmailOnEventTrigger.cs* in an editor. Copy its content to the created *myapp/SendEmailOnEventTrigger.cs*

Open the *myapp/SendEmailOnEventTrigger.cs* file and check it.

When a batch of new messages is received from an event hub, the host attempts to trigger the function calling the *Run()* method. Each event hub message will be sent to an email account. If you look at the *SendEmail()* method, you will see that we are sending an email to a Gmail account:

using (var client = new SmtpClient())

{

client.UseDefaultCredentials = false;

client.Credentials = new System.Net.NetworkCredential(username, password);

**client.Port = 587;**

**client.Host = "smtp.gmail.com";**

client.DeliveryMethod = SmtpDeliveryMethod.Network;

client.EnableSsl = true;

await client.SendMailAsync(msg);

}

Personally, I have a Gmail account and want to use it to test this function app. If you prefer to use another mail server – modify this code accordingly.

The last thing to do is to build and publish the function.

1. Open a Command Prompt window (or a Terminal window in the Visual Studio Code)
2. Change the current working directory to the *myapp* (or other directory, containing your new function app)
3. Run the code to build the project:

**> dotnet restore**

**> dotnet build**

1. If there are no errors, click **Deploy to Function App…** icon to publish it to the cloud

Graphical user interface, text, application

Description automatically generated

1. Select Function App in Azure **anomalyeventtrigger** to deploy the trigger
2. When deployment is successfully completed, open **anomalyeventtrigger** in the portal and select **Functions** tab

Graphical user interface, text, application, email

Description automatically generated

Our **anomalyeventtrigger** should appear in the list of functions. But if you browsed the code, you could notice some settings, not defined anywhere: **EMAIL\_USER\_NAME**, **EMAIL\_PASSWORD**, and **EVENTHUB\_Connection**.

The **EVENTHUB\_Connection** is a connection string for access to your event hub. To find it:

1. Open **anomaly-detection-rg** resource group in the portal
2. Find the Event Hub Namespace containing hubs **eventhub1** and **eventhub2**
3. Select **Shared access policies** tab, click **eventHubSharedAccessPolicy** and copy **Connection string–primary key**

Graphical user interface, text, application, email

Description automatically generated

This **Connection string** is your **EVENTHUB\_Connection** value.

The **EMAIL\_USER\_NAME** will be used as an email recipient and sender at the same time. That can be your Gmail user account (e.g., “*myemail@gmail.com*”).

The **EMAIL\_PASSWORD** should be a password, generated by Google, for sending emails from applications using your Gmail account (not your user email password!). Read the article [Sign in with App Passwords](https://support.google.com/accounts/answer/185833?hl=en) how to generate the password. When you generate a password, save it somewhere in your notes.

The last thing to do is to configure these settings in the portal.

1. Open **anomalyeventtrigger** in the portal and select **Configuration** tab
2. Click **+ New application setting**

Graphical user interface, application

Description automatically generated

1. Enter ***EMAIL\_USER\_NAME*** as a **Name**, an email address as a **Value** and click **OK**
2. Enter ***EMAIL\_PASSWORD*** as a **Name**, a generated password as a **Value** and click **OK**
3. Enter ***EVENTHUB\_Connection*** as a **Name**, a connection string as a **Value** and click **OK**
4. Do not forget to click **Save** to save your changes

The result should be like what is shown below:

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

### 11.4 Test Event Hub Trigger

Now everything is ready to check how our event hub trigger is working.

1. Run your IoT device simulator (see section [Run IoT device simulator](#_4.2.2_Run_IoT)) and leave it running for a few minutes
2. Open **anomalyeventtrigger** in the portal and select **Functions** tab
3. Click **SendEmailOnEventTrigger** function
4. Select **Monitor** tab. You will see recent logs:

Graphical user interface, application

Description automatically generated

Click on any log record to see details:

Graphical user interface, text, application, letter

Description automatically generated

The log contains an event message received by the event hub trigger like this one:

[{"device-id":"device001","time":"2021-12-22T12:40:21.4170000Z","temperature":20.28,"temperature\_anomaly":1,"temperature\_anomaly\_score":1E-08,"first":1}]

The message contains:

* Device id
* Temperature
* *temperature\_anomaly* flag, indicating the temperature is anomalous
* *temperature\_anomaly\_score*
* *first* flag, indicating that this event is first in the 60-seconds sliding time frame.

You can use the *first* flag to check if a message with anomalous temperature is not first (*“first”: 0*) in a series of events, occurred during last 60 seconds. If is not first, you can skip it not sending emails, because you have already sent an email on receiving a “first” event.

Try to increase or decrease temperature, the IoT device simulator is sending (see [Run IoT device simulator](#_4.2.2_Run_IoT)). Open your email client and check emails with the subject “Temperature anomaly detected”, sent to the email account, specified as the **EMAIL\_USER\_NAME**. The emails will contain messages, received by the Event Hub Trigger.

# Summary

# 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)
15. [Azure Time Series Insights Explorer](https://docs.microsoft.com/en-us/azure/time-series-insights/concepts-ux-panels)
16. [Visualize data from Azure Time Series Insights in Power BI](https://docs.microsoft.com/en-us/azure/time-series-insights/how-to-connect-power-bi)
17. [Azure IoT CLI](https://docs.microsoft.com/en-us/cli/azure/iot?view=azure-cli-latest)
18. [Azure CLI reference commands for Azure IoT](https://docs.microsoft.com/en-us/cli/azure/azure-cli-reference-for-iot)
19. [Install Azure CLI on Windows](https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-windows?tabs=azure-cli)
20. [Microsoft Azure IoT extension for Azure CLI](https://github.com/Azure/azure-iot-cli-extension)
21. [Installing PowerShell on Windows](https://docs.microsoft.com/en-us/powershell/scripting/install/installing-powershell-on-windows?view=powershell-7.2)
22. [PowerShell Language Support for Visual Studio Code](https://marketplace.visualstudio.com/items?itemName=ms-vscode.PowerShell)
23. [Sign in with Azure CLI](https://docs.microsoft.com/en-us/cli/azure/authenticate-azure-cli)
24. [Develop Azure Functions by using Visual Studio Code](https://docs.microsoft.com/en-us/azure/azure-functions/functions-develop-vs-code?tabs=csharp)
25. [Downloading and installing Node.js and npm](https://docs.npmjs.com/downloading-and-installing-node-js-and-npm)
26. [Azure Functions for Visual Studio Code](https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-azurefunctions)
27. [Azure Event Hubs trigger for Azure Functions](https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-event-hubs-trigger?tabs=csharp)
28. [Sign in with App Passwords](https://support.google.com/accounts/answer/185833?hl=en)