Azure IoT Hub Cloud Connection

□

Contents

- 1 Introduction
- 2 Prerequisites
- 3 Azure account creation
- 4 Managing Azure services
- 5 Configuring Azure IoT Hub on RutOS
  - 5.1 SAS key connection type configuration
  - 5.2 Device Provisioning Service (DPS) configuration
    - 5.2.1 DPS X.509 attestation
    - 5.2.2 DPS Symmetric key mechanism
- 6 Direct methods configuration
  - 6.1 Using internal device API with Direct Methods
  - 6.2 IoT plug and Play configuration
- 7 Sending data with "Data to Server" feature
  - 7.1 Data to Server configuration
    - 7.1.1 Checking if data reaches IoT Hub on Azure
- 8 Example use cases
  - 8.1 Dynamically monitoring Modbus data
    - 8.1.1 Modbus TCP Server
    - 8.1.2 Modbus TCP Client
    - 8.1.3 Data to Server
    - 8.1.4 Monitoring and controlling incoming data
- 9 External links

Introduction

This article contains instructions on how to configure a Teltonika Networks device in order to connect to the Azure IoT Hub. **Azure IoT Hub** is a managed service hosted in the cloud that acts as a central message hub for communication between an IoT application and its attached devices.

The information in this page is updated in accordance with the RutOS 7.8 firmware version.

For information on Azure services utilizing legacy firmware, please click here.

For a wiki on Azure services based on RutOS versions prior to the 7.8 update, please click here.

Prerequisites

You will need:

- A Teltonika Networks device;
- An Azure IoT Hub account.

Azure account creation

Visit https://azure.microsoft.com/en-us/ and create an account that will suit your needs, for testing purposes we will be using free Azure account.

Managing Azure services

- First you will want to create a Resource group for easier management of resources that will be added later. In Microsoft Azure home page.
  Select **Resource groups**
  If it is not in very first page, click **More services** and locate it there.
  
- In new window, select **Add**
  
- And finish creating the Resource group
  Select your subscription, for this example **Free Trial** will be used.
  1. Name your group
  2. Choose server location for meta data. We will choose ***(South America) Brazil South*** and will use it during this example.

  
- At this moment Tags will be skipped, press **Review + create** at the bottom of screen and click Create to finish setup.
  
  
- After being redirected to Homepage click on **Resource groups**. You should see the newly created group, select it and press **Add**.
  
- Select **Internet of Things** or simply search **IoT Hub** and press **Create**.
  
- We leave default subscription and resource group and choose:
  1. Region – **(South America) Brazil South** as before
  2. Create a name for IoT Hub
  3. Go to **Size and scale tab**

  
- For testing purposes, we are using **F1: Free tier**
  At the bottom of the screen **Review + create**

  

  Click on **>> Create**

  

  Note: Wait until resource deploys and press Go to **Resources**

- Inside IoT Hub list:
  Scroll down to Explorers and select **IoT devices**

  
- Press **New**
  

- In new device creation
  1. Enter Device ID
  2. Leave everything else on default and press **Save**

  
- After creating a new device you will be redirected back to IoT devices. Select the newly created **Device ID**

  
- In the device window you will find information needed to connect Teltonika devices to Azure IoT Hub.
  For now only **connection string** will be used. Copy Primary Connection string.

  

Configuring Azure IoT Hub on RutOS

To configure an Azure IoT Hub instance on the Teltonika device it is essential to install the Azure IoT Hub package via the package manager.

- To install required package navigate to **System > Package Manager** and install Azure IoT Hub package

  

Navigate to **Services > Cloud solutions > Azure IoT Hub** and add a new instance. In the pop-up window there will be two different connection types available:

- **Shared Access signature (SAS) key**
- **Device Provisioning Service (DPS)**

In this article both connection types will be demonstrated.

SAS key connection type configuration

Configuring Azure IoT Hub using the SAS key connection type is straightforward:

1. Make sure to enable the instance by pressing **"Enable"** button
2. Paste previously copied **"Primary Connection String"**
3. Press **"Save & Apply"** button

After the instance is correctly configured the connection status icon will be visible. A green dot indicates that the connection is successful. 

Device Provisioning Service (DPS) configuration

The IoT Hub Device Provisioning Service (DPS) is a helper service for IoT Hub that enables zero-touch, just-in-time provisioning to the right IoT hub without requiring human intervention, allowing customers to provision millions of devices in a secure and scalable manner.

One of the primary features of DPS is its capability to dynamically manage multiple device identities. This service manages the device identity creation using enrollments which can be configured using the following attestation types:

- 1. X.509 intermediate certificates
- 2. Symmetric keys

To learn more about DPS service read about it here

DPS X.509 attestation

1. The initial step generating certificates. Refer to the Microsoft guide, section *"Create an X.509 certificate chain"* for in depth information that explains each step of the process in detail.

The required certificates and keys:

1. **Root CA certificate**
2. **Intermediate CA certificate**
3. **Devices certificates**

2. After successfully generating the certificates return to the Azure portal page and navigate to your Azure IoT Hub Device Provisioning Service (DPS) page. From there proceed create an enrollment group. Refer to the Microsoft guide, section *"Create an enrollment group"* for in depth information.

3. Return to the device WebUI and navigate to **Services -> Cloud Solutions -> Azure IoT Hub** page to create a new configuration instance:

3.1 Set connection type as a **Device Provisioning Service (DPS)**;

3.2 Enter **"ID Scope"** of your DPS service page on Azure. This value can be retrieved from the DPS instance found on Azure Portal page or by following the earlier guide;

3.3 Specify the **"Registration ID"**. This is the subject common name (CN) of the device leaf certificate that was created using the earlier guide.

3.4 Upload the certificate chain file and the private key file.

With all the required values in place the configuration pop-up window should resemble the screenshot below:

After a couple of moments the status of the configured instance status icon should turn green indicating the device has successfully established connection to Azure server.

DPS Symmetric key mechanism

The Symmetric key mechanism configuration is more straightforward. To configure it go back to the Azure portal page, navigate to your DPS service page and create a new enrollment group with the Symmetric key attestation mechanism.

Inspecting the newly created enrollment group will reveal some keys. The primary key will be used to derive each individual device identity. This can be done using a script which is

available on Microsoft DPS documentation here.

The script contains a couple of variables: **"KEY"** and **"REG_ID"**. The KEY field shall contain the primary key which is obtained from the newly created enrollment group. The **"REG_ID"** field specifies the device identity name that will be created. Upon executing the script a shared access key will be created.

Go back to the device WebUI **Services -> Cloud Solutions -> Azure IoT Hub** configuration page and add a new instance. In the configuration window select DPS connection type and Symmetric Key connection type.

- In the **"ID scope"** field specify your Azure DPS service ID. This value can be retrieved from the DPS instance found on Azure Portal page or by following the earlier guides from Microsoft.
- In the **"Registration ID"** field enter the **"REG_ID"** value that was specified in the earlier script.
- In the **"Symmetric key"** field enter the output value of the script that was used earlier].

If you are following this guide your configuration window should look similar to the screenshot below.

After a few moments the device should establish connection to the Azure server.

After returning to the IoT Hub services in the Azure portal page it can be observed that the DPS service has created a new device identity was named the same as what we specified in the **"REG_ID"** field in the script.

Direct methods configuration

IoT Hub direct methods enable you to remotely invoke calls on devices from the cloud. Direct methods follow a request-response pattern and are meant for communications that require immediate confirmation of their result.

By default, all configuration instances will have this option disabled. To enable it, navigate on the router WebUI to **Services -> Cloud Solutions -> Azure IoT Hub** and press the edit button on the specific instance. Afterwards the "**Enable Direct Methods**" option needs to be enabled.

For testing and demonstration purposes we will use the Azure IoT Explorer application. The Azure IoT Explorer is a graphical tool for interacting with devices connected to your IoT hub. To learn more about this tool you can follow the Microsoft documentation here.

Using internal device API with Direct Methods

After enabling the direct method feature go to Azure IoT Explorer, select the appropriate device identity and navigate to the direct methods tab. All our devices support the **"api_call"** direct method

which exposes the API interface to be used from the Azure side. In this example we will make a simple GET request to retrieve the I/O status of the device. Full documentation of Teltonika devices API can be found here .

The Azure IoT Explorer direct method tab will have a Payload field. In this field the **"api_call"** method expects JSON formatted data.

The API call expects at least two parameters. The first one is called **"method"** which needs to have an integer value between zero and three, corresponding to the API method type - either GET, POST, PUT or DELETE. The second parameter is **"endpoint"** which expects a string value of the API endpoint. In this case, we will call the **"/io/status"** endpoint.

After pressing the "**Invoke Method**" button the response from the device will be visible which is a standard API response specified in our documentation.

To determine the appropriate payload and method to use we provide an additional file currently called **"teltonikaGenericDevice.json"**. This file is written in **Digital Twin Definition Language (DTDL)**. To learn more about DTDLs and Digital Twins read about it in Microsoft documentation here.

In this file you can see that it supports the api_call method, which accepts three values. The request body is optional, as some methods, such as the GET method, may not require it. JSON files can be downloaded here.

The IoT Explorer can be configured to parse DTDL files and display them to the user for easier work.

IoT plug and Play configuration

Navigate to the "**IoT Plug and Play components**" tab on the IoT Explorer. Initially there may be an error stating that it did not retrieve an interface model. To resolve this click on the "**Configure**" button. In this guide a local folder will be added by pressing the "**Add**" button.

The specified directory must have the DTDL **".json"** files. After adding the local folder press the **"Save"** button.

Return to the device identity Plug and Play tab. Now you will be able to see two components with model IDs named "**genericDevice**" and "**deviceInformation**". The generic device will display the DTDL interface description.

In the upper toolbar select the "**Commands**" tab. There you will see that IoT Explorer has parsed the API call method and created three new fields. Now we can try to call the same I/O status method that we called previously.

We can see that some information was correctly retrieved from the router. This workflow makes it easier to work with API calls from the Azure side.

Sending data with "Data to Server" feature

The Data to Server service allows you to set up collections that collect data from various sources and periodically sends it to various servers. We can configure this feature to send data from the device to Azure IoT Hub.

Data to Server configuration

To configure the **Data to Server** service on Teltonika devices navigate to **Services -> Data to Server** on the WebUI. This guide will cover only the collection output part. To learn more about Data to Server features you can find the dedicated guide on our Wiki. From this point it is assumed the collection is properly set up with correct inputs.

1. In the server configuration configuration window, choose "**Type**" as a "**Azure IoT Hub**" option;

   In the "Configuration type" field you can choose whether to use an existing Azure IoT Hub configuration or configure a new and unique Azure IoT Hub configuration. In this example we will stick with the previously created Azure IoT Hub instance configuration. If you choose to create a new unique Azure IoT Hub configuration on the Data to Server instance you will need to input all the options that were discussed earlier in this guide.

2. Select your preferred Azure IoT Hub instance;
3. Press "**Save & Apply**" button.

   If you are following this guide your configuration should resemble something similar to the screenshot below.

Checking if data reaches IoT Hub on Azure

To determine whether data successfully reaches Azure IoT Hub select your device and navigate to the "Telemetry" tab on the Azure IoT Explorer. Ensure that **"Use built-in event hub"** option is enabled and press the **"Start"** button. After some time you should see that data was sent from the device to the Azure IoT Hub.

Example use cases

This section shows some practical examples that combines most of the features that are discussed earlier.

Dynamically monitoring Modbus data

This example shows how to monitor and dynamically control Modbus data purely from the Azure cloud. We will set up a simple Modbus TCP server and client with Data to Server which will forward all the incoming Modbus data to Azure IoT Hub. In the end using direct methods we will showcase how Modbus TCP client can collect and report different data from the cloud. For this we will be using the following services:

- Azure IoT Hub;
- Data to Server;
- Modbus TCP Server;
- Modbus TCP Client;
- An Azure IoT Hub account.

In this example the Azure IoT Hub WebUI service configuration will not be covered since all the necessary information can be found in the earlier sections.

Modbus TCP Server

Enable the service in **Services -> Modbus -> Modbus TCP Server** with **"Enable"** option. For more information about this service you can find it on our Modbus Wiki TCP server section

Modbus TCP Client

Go to **Services -> Modbus -> Modbus TCP Client** page and create a new instance. This part will assume most of the configuration for this page is already made. For more information about this service you can find it on our Modbus Wiki TCP client section.

For this use case a single Modbus request configuration will be created to request the current device timestamp:

- Data type: **32bit UINT, Byte order 1,2,3,4**
- Function: **Read holding registers (3)**
- First register number: **2**
- Register counter/Values: **2**

For device specific Modbus register information refer to the appropriate Wiki documentation.

Data to Server

1. Go to **Services -> Data to Server** page and create a new collection instance.

1.1. Select the input **Type** to **"Modbus"**;

1.2. Change the **Format type** to **"Custom"**;

1.3. In the **Format string** we will enter the following data:

```
{"Date (Linux timestamp)": %timestamp%, "MODBUS server ID": "%server_id%",
"MODBUS server name" : "%server_name%", "Request name": "%name%", "Start
register": "%addr%", "Register data (JSON object)": %data%, "Raw data":
"%raw_data%"}
```

This will form requests about Modbus data including the register values;

1.4. In the **"Collection configuration"** page select the **"Format type"** to **"Custom"**;

1.5. Change the **"Format string"** to **{ "input1": %input1% }**. Make sure to change the **%input1%** value to your specific input name. Note that this value is not enclosed in braces. This is intentional since the braces are present in the Modbus input **Format string** field;

1.6. In the **"Server configuration"** select the **"Type"** to **"Azure IoT Hub"** and configure the Azure

configuration instance in accordance to your needs.

Monitoring and controlling incoming data

Inspecting the incoming data to the Azure IoT Hub using Azure IoT Explorer reveals that Modbus data is being received successfully.

In order to change the type of Modbus data sent to the Azure IoT Hub without going to the device WebUI the **Direct Methods** feature can be utilized. Using Azure IoT Explorer go to the device identity that was configured on the Teltonika device and select **"Direct method"** tab.

Using the **"api_call"** direct method create API requests that update the Modbus TCP Client request configurations (API reference for Modbus services can be found here. In this example the request configuration will be changed to collect **"Mobile signal strength"**. To do this using only API we will need to resolve the Modbus TCP client instance ID then the request ID of the instance which currently collects the temperature data. Using both of the IDs we will form the last API PUT request to update the register values:

2.1. Invoke **"/modbus/client/tcp/config"** GET request and inspect the output on IoT Explorer. The **"data"** array will contain JSON objects of every configured client instance. The **"id"** value will be used when forming the next API request;

2.2. Invoke **"/modbus/client/tcp/{id}/requests/config"** GET request and replace the **"{id}"** with the **"id"** value from the previous step. Inspecting the output will reveal the **"data"** array which contain JSON objects of every configured request. The **"id"** value of the request that collects temperature data will be used when forming the next API request;

2.3. Invoke **"/modbus/client/tcp/{id}/requests/config/{request_id}"** PUT request and replace the **"{id}"** with the **"id"** value from the 2.1. step and replace the **"{request_id}"** with the **"id"** value from the 2.2. step. In the **"request_body"** field add values to change the first register and data type values: **{"data":{"first_reg":"4","data_type":"32bit_int1234"}}** which corresponds to mobile signal strength register (other parameters such as number of registers stay the same since the timestamp register count is the same too).

2.4. Observe the incoming telemetry data. At this point the register data shall contain the mobile signal strength values.

This concludes the Azure IoT Hub guide. Using direct methods with device API is a powerful tool that unlocks new capabilities of device monitoring and control from the cloud. Nearly all features available via device WebUI are also available using API.

External links

1. https://azure.microsoft.com/en-us/
2. https://developers.teltonika-networks.com/
3. https://learn.microsoft.com/en-us/azure/iot/howto-use-iot-explorer
4. https://learn.microsoft.com/en-us/azure/iot-dps/
5. https://learn.microsoft.com/en-us/azure/iot-dps/how-to-legacy-device-symm-key?tabs=linux&amp%3Bpivots=programming-language-ansi-c&pivots=programming-language-ansi-c#derive-a-device-key
6. https://learn.microsoft.com/en-us/azure/iot-dps/tutorial-custom-hsm-enrollment-group-x509?pivots=programming-language-ansi-c#create-an-x509-certificate-chain
7. https://learn.microsoft.com/en-us/azure/iot-dps/tutorial-custom-hsm-enrollment-group-x509?pivots=programming-language-ansi-c#create-an-enrollment-group
8. https://developers.teltonika-networks.com/reference/7.6.10/v1/modbus/
9. https://wiki.teltonika-networks.com/view/RUTX11_Modbus#Modbus_TCP_Client
10. https://wiki.teltonika-networks.com/view/RUTX11_Modbus#Modbus_TCP_Server
11. https://learn.microsoft.com/en-us/azure/digital-twins/concepts-models
12. https://developers.teltonika-networks.com/fundamentals/#request-and-response-structures