FAQ COMBIVIS Connect

MQTT Cloud Router with Microsoft Azure

Part	Version	Revision	Date	Status
en	5.1.1183.1	001	2021-05-03	Released

Content

Introduction

This document explains how to push machine data to an Azure IoT Hub and store it in Azure Cosmos DB.

The following are required for this tutorial:

- C6 Router with Cloud Runtime
- Combivis Connect Domain
- Combivis Studio HMI
- Microsoft Azure Account

Azure Setup

This section explains the setup of Azure services to handle the incoming binary MQTT data, translate it to JSON format, and store it in a database.

1. Create a Resource Group

Fig. 1: Azure Resource Group Icon

Resource groups organize the Azure services used for a particular solution. When configuring Azure resources in the following steps, add them to this resource group.

To create a resource group, navigate to the portal menu icon located on the left side of the Azure portal toolbar.

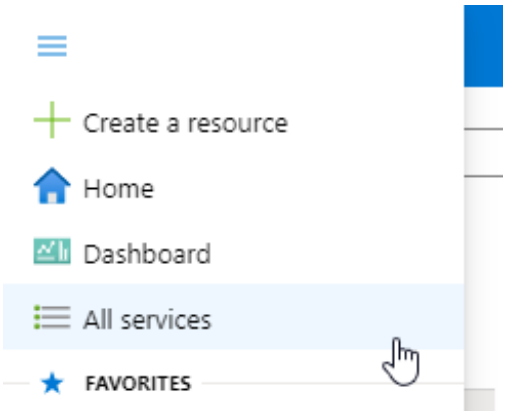

Fig. 2: All Services in Toolbar

Select *All Services* > *All* > *Resource Groups*. Select *+ Add*, fill in the required fields, select *Review + Create*, then *Create*.

The resources created in the following steps may all be found under *All Services* > *Internet of Things*. To create a resource, select the resource, then select *+ Add* and fill in all required fields.

2. Create an IoT Hub

Fig. 3: Azure IoT Hub Icon

IoT Hubs act as MQTT brokers for IoT devices. The name of this and other Azure services must be globally unique. The status of deployments may be viewed at any time in the notifications tab which may be accessed from the toolbar at the top of the page.

Fig. 4: Notifications in Azure Toolbar

Add an IoT device to the IoT Hub by navigating to the *IoT Hub* > *IoT Devices* and select *+ New*. This IoT device corresponds to the KEB cloud router. Select *Symmetric key* for *Authentication type* and check *Auto-generate keys* when setting up this device.

An SAS token must now be generated to allow connection between the router and the IoT Hub. To do this, select the Cloud Shell icon located on the portal toolbar.

Fig. 5: Cloud Shell in Azure Toolbar

Paste the following command into the Cloud Shell window, replacing myresourcegroup and myiothub with the names of the resource group and IoT Hub, respectively (keep the double-quotes):

New-AzIotHubSasToken -ResourceGroupName “myresourcegroup” -IotHubName “myiothub”

Hit enter. Save the SaS token that Cloud Shell generates in the following line. This is necessary for configuration of the router. See the example below with a resource group named *test-resource-group* and an IoT Hub named *test-iot-hub-keb*.

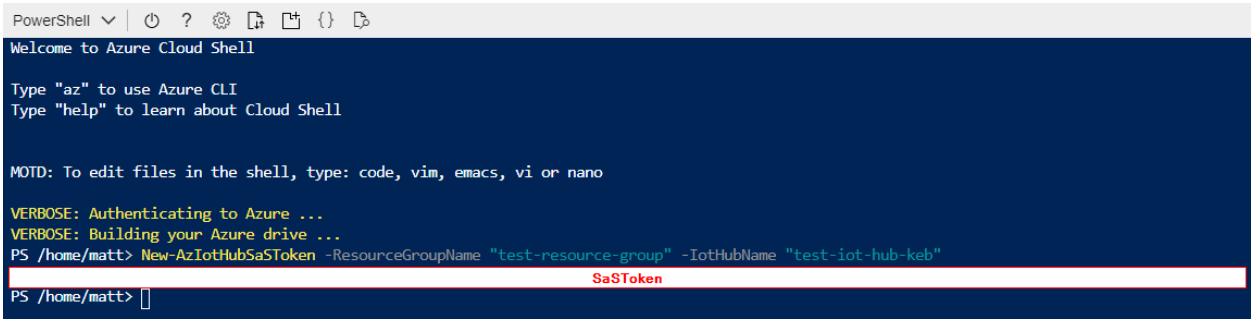

Fig. 6: SaS Token Notifications in Cloud Shell

3. Create a Cosmos DB account

Fig. 7: Azure Cosmos DB Icon

Create a Cosmos DB account. Select *Core (SQL)* for the database API (Application Programming Interface) when creating this account. SQL is the only API that is supported by the Azure Function service bindings that will be developed in the next step.

Add a container to the Cosmos DB account by navigating to the Cosmos DB account > *Data Explorer* and select *New Container*. Check *Create new Database id* and assign a name to the *Database id* and *Container id*. Make note of these names for use in the next step. Fill in the *Partition key* field with a parameter that you would like to sort data by. For the cloud router messages, partition key options include: /deviceId, /tag, /value, /type, /timestamp, and /timezone. These five parameters exist for every tag value that is pushed from the router. The structure of the created SQL Cosmos DB is shown below. The translated data will be found in the container under *Items* in JSON format.

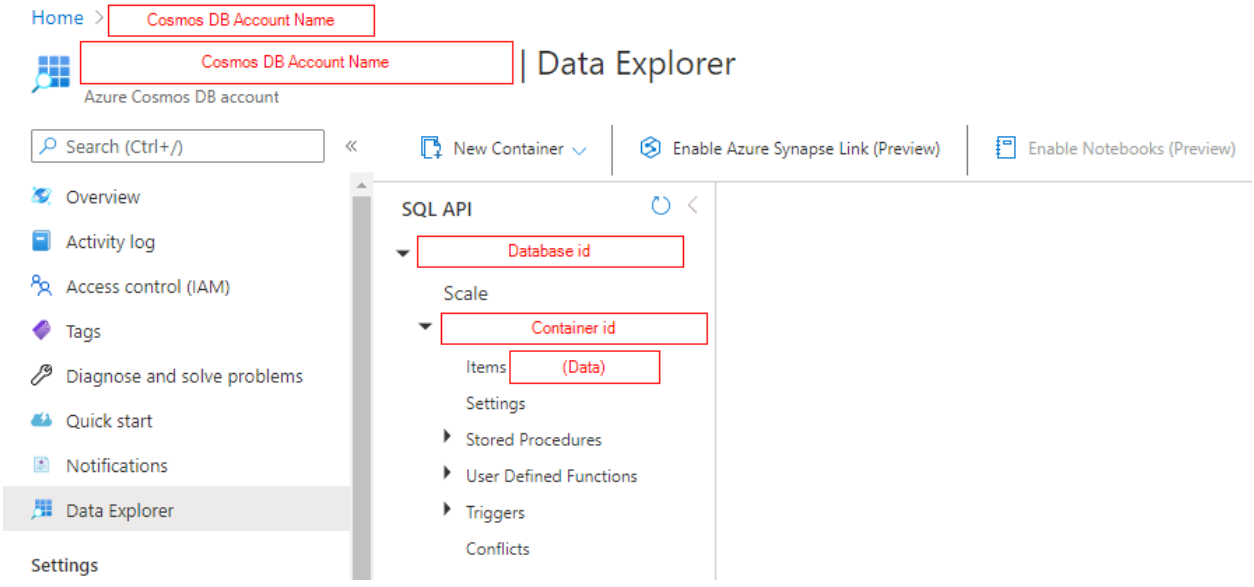

Fig. 8: Structure of Cosmos DB with SQL API

4. Create a Function App

Fig. 9: Azure Function App Icon

KEB's cloud routers send messages to the cloud in binary format. In order to store data, these binary messages received by the IoT Hub must be sent to an Azure Function to be decoded before being stored in Cosmos DB. Assign a name to the Function App, select *Publish* > *Code*, select *Node.js* for the *Runtime stack*, and choose *Version 12 LTS*.

Add a Function to the Function App by navigating to the Function App > *Functions* and select *+ Add*. Choose *Develop in portal* and select the *IoT Hub (Event Hub)* template. Assign a name to the function, under *Event Hub connection* select *New* > *IoT Hub*, then select your IoT Hub under *Event Hub Connection*. Select binary for *Data type of the trigger input* since the cloud router sends binary packets to the IoT Hub. Keep all other default values. Navigate to the *Integration* tab of the newly created function. From the template, the function now triggers with each new binary packet received by the IoT Hub from the router. This is referred to as an "input binding" by Azure. Next, an output binding for the Cosmos DB storage must be created. Select *+ Add output*. For *Binding Type*, select *Azure Cosmos DB*, insert the name of your Cosmos DB Database id from the previous step under *Database name* and place your Container id in the *Collection Name* field. For Cosmos DB account connection select *New* > *Azure Cosmos DB Account* and choose your account from the drop-down menu.

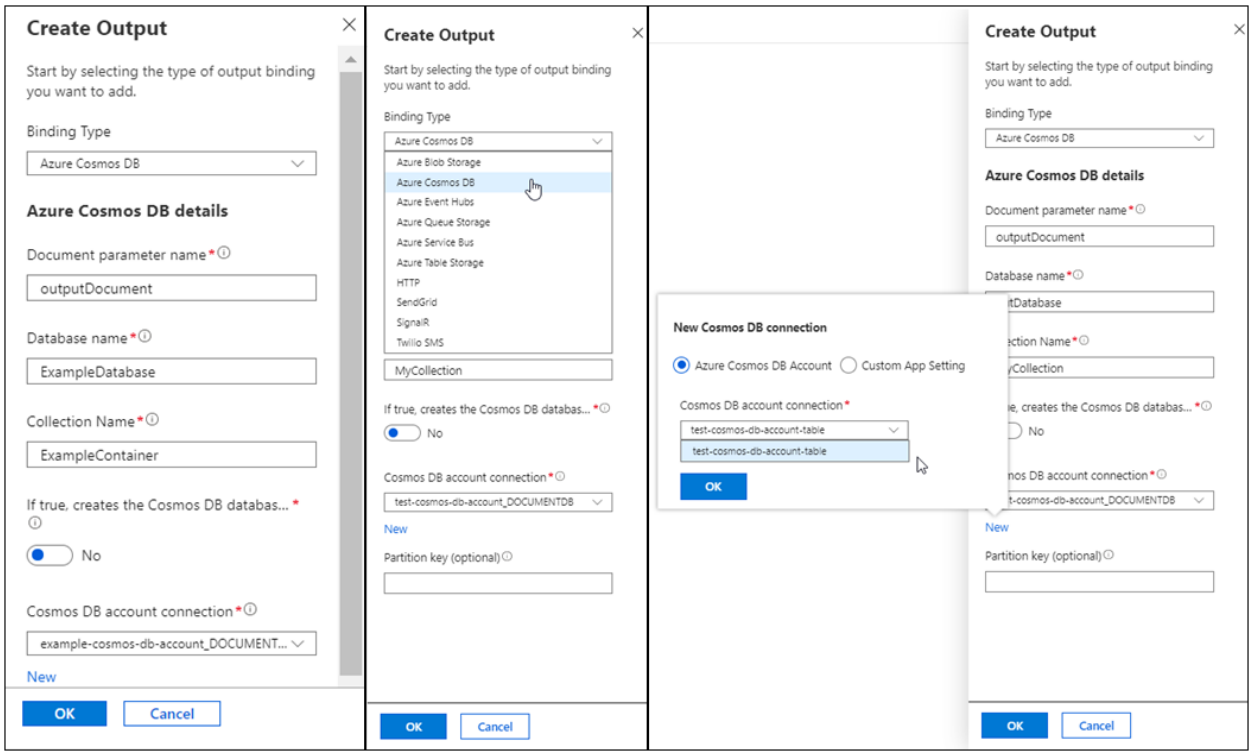

Fig. 10: Output Binding Settings

Leave all other fields in this window as their default. Note: failure to leave the *Event parameter name* on the input binding and *Document parameter name* on the output binding as their default values will result in a function error for the code provided. These parameter names refer to parameters used in the code.

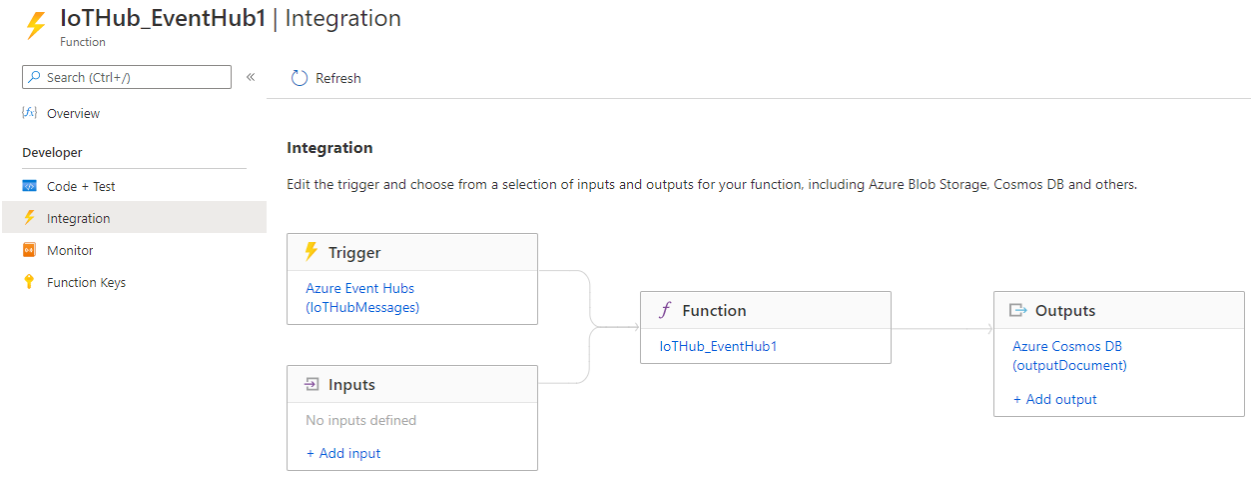

Fig. 11: Layout of Completed Azure Function

Finally, go to the *Code + Test* tab of the Function. Remove any auto-populated code in the index.js file shown and paste in the code from the text file supplied with this tutorial. Note: do not change the function.json file. The function app is now complete and will trigger with each IoT message, decode the message, and output it to Azure Cosmos DB where it is stored and may be further processed if desired.

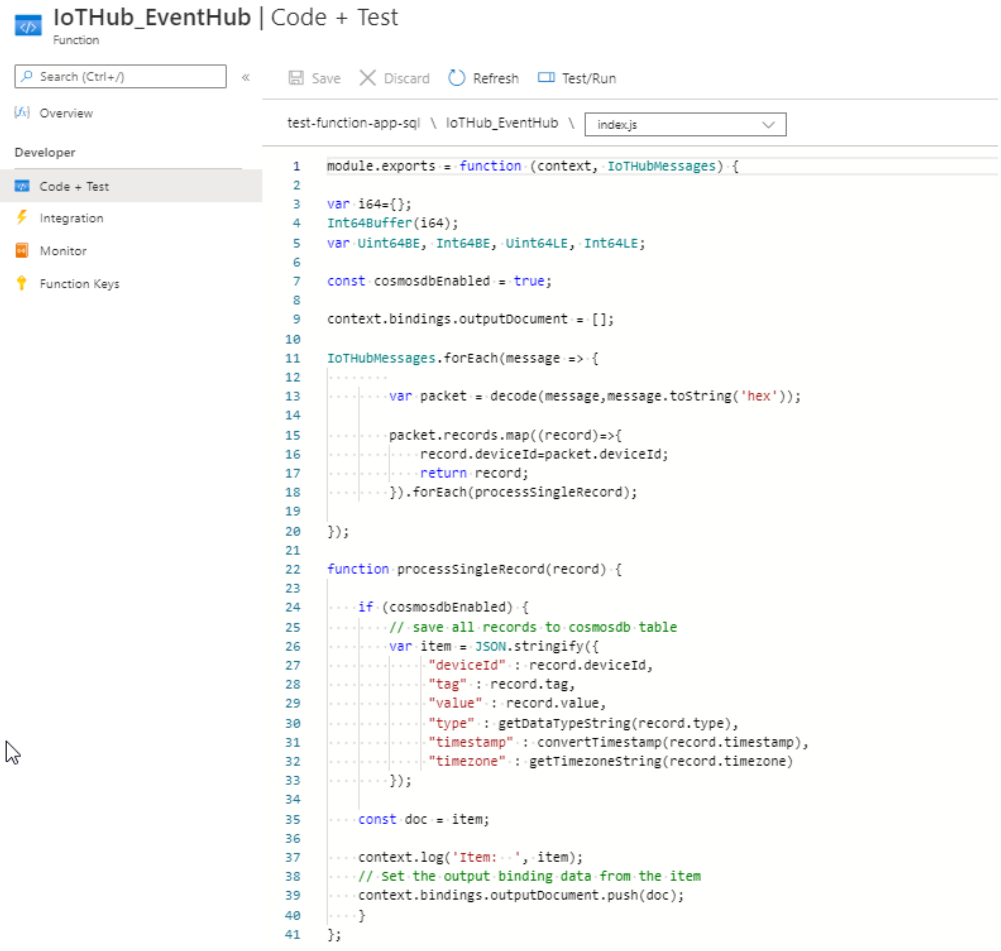

Fig. 12: Function App Code + Test Tab

Router Configuration

This section explains how to configure a cloud router for MQTT communication via Studio HMI.

1. Create a Studio HMI Project

Add a Data Logger to the HMI project. The Data Logger resource is used to push data to the cloud. Tags are configured to the data logger and recorded into the database.

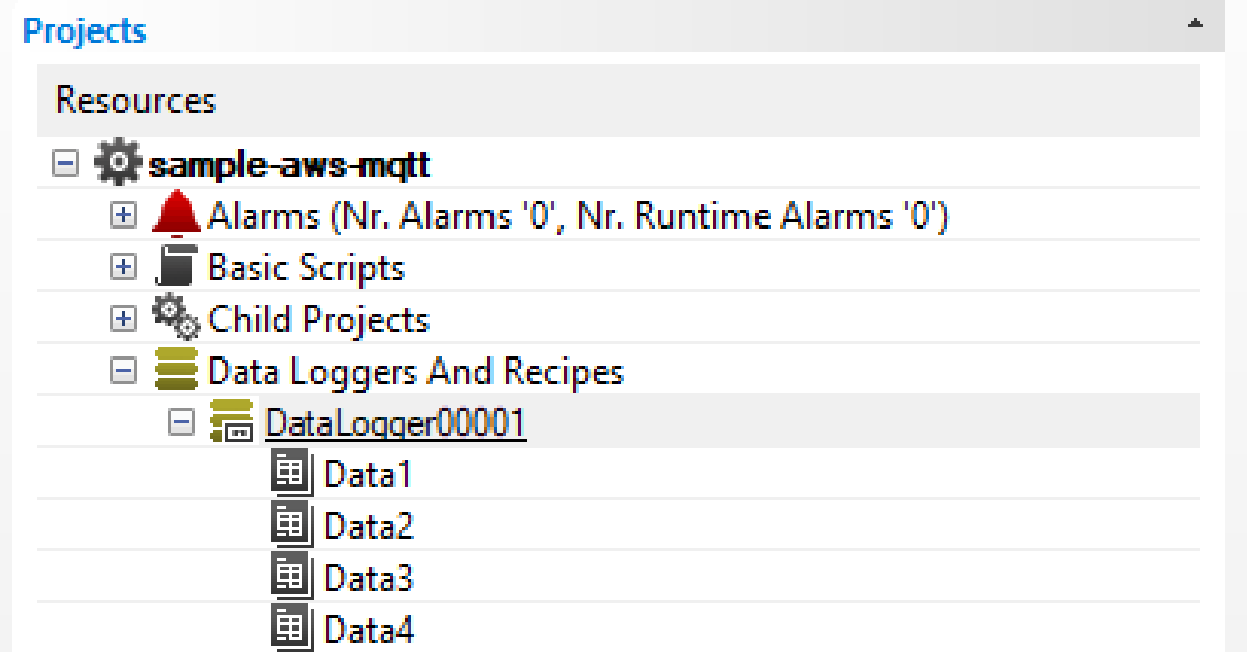

Fig. 13: Data Logger

The *Style* property defines how the data logger records entries into the database. If the setting *Record on Change* is selected, the datalogger will add entries into the database any time a tag within the datalogger changes its value. The setting *Record on Command* adds entries into the database any time the *Recording Variable* is strobed up in the *Execution* properties. This is the style of the date logger in the sample HMI project supplied with this tutorial.

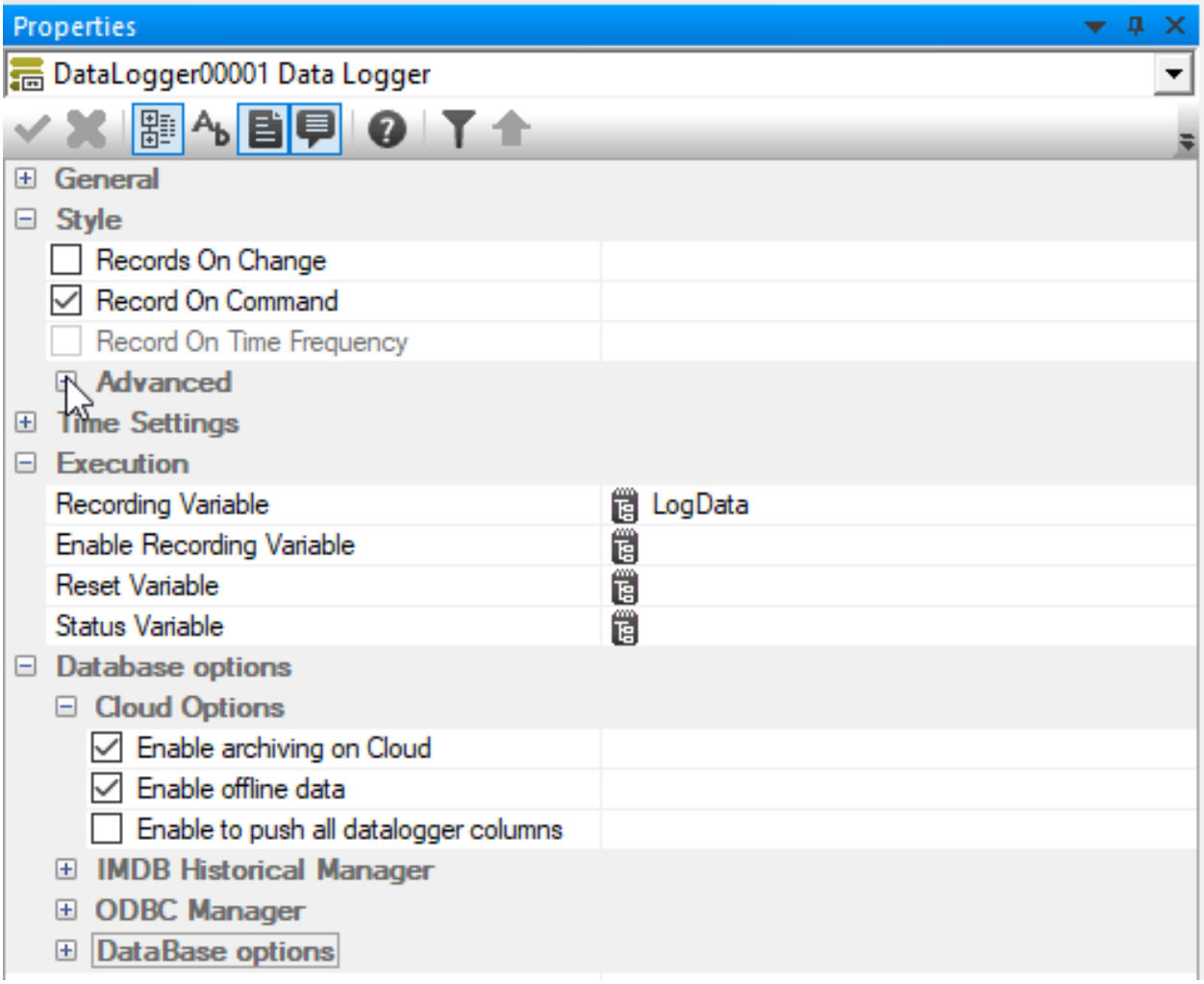

Fig. 14: Data Logger Configuration

Lastly, the *Cloud Options* need to be configured in the *Database options* properties. The *Enable archiving on Cloud* property is required to push data to the cloud. The *Enable offline data* propery will save data locally to the router in the event of internet loss.

The frequency in which the router will push data to the cloud is configured using the *Publish Period (ms)* setting in the *Cloud Push Agent Configuration* properties.

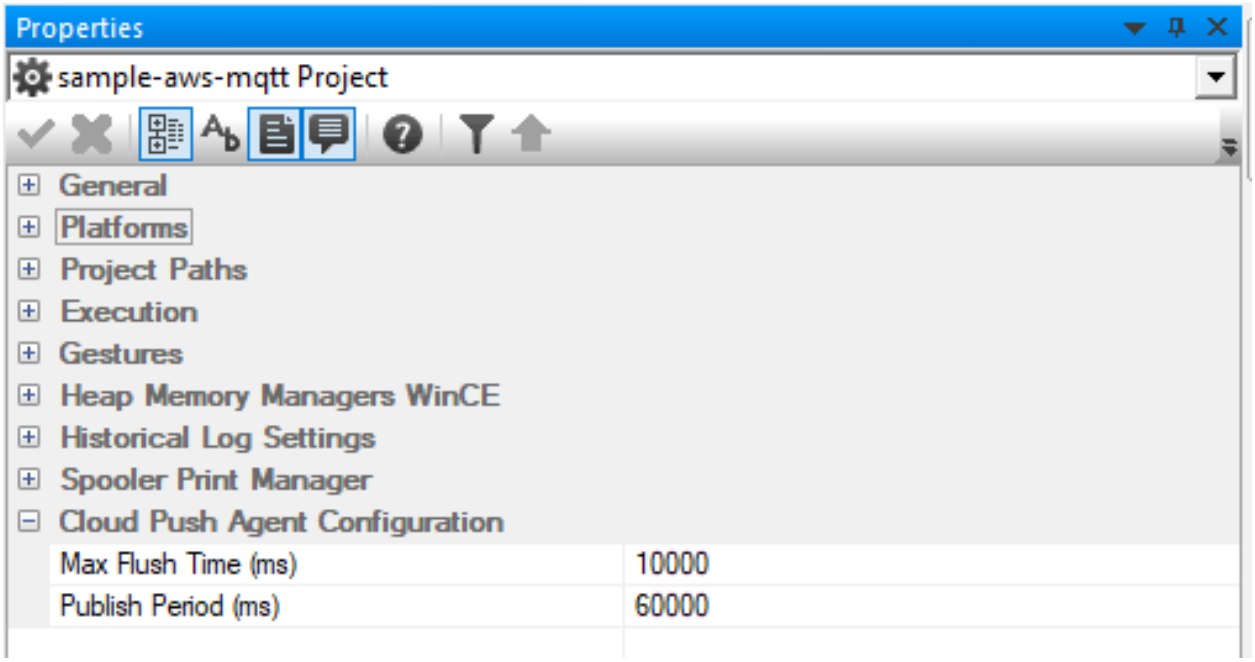

Fig. 15: Cloud push agent configuration

2. Edit Cloud Configuration File

When a Studio HMI project is created, a cloud runtime configuration XML file is created in the *Resources* folder of the project directory named *cloudruntimeconfig.xml*. Edit this XML file with a text editor (Notepad++ is recommended). Uncomment the lines shown below under *<ManualConfigOptions>*.

```
<?xml version="1.0" encoding="UTF-8"?>
<CloudRuntimeConfiguration>
    <Version>1</Version>
    <!-- <UseManualAuthenticationMode>true</UseManualAuthenticationMode> -->
    <!-- <CloudRuntimeDataDirectory></CloudRuntimeDataDirectory> -->
    <!-- <CloudRuntimeDataDirectoryHidden>false</CloudRuntimeDataDirectoryHidden> -->
    <!-- <CloudRuntimeDataFilesHidden>false</CloudRuntimeDataFilesHidden> -->
    <ManualConfigOptions>
        <DeviceID></DeviceID>
        <ClientID></ClientID>
        <Endpoint></Endpoint>
        <Password></Password>
        <Protocol>mqtt</Protocol>
        <Qos>1</Qos>
        <Target></Target>
        <!-- <TrustStoreFilePath></TrustStoreFilePath> -->
        <!-- <ClientCertificateFilePath></ClientCertificateFilePath> -->
        <!-- <ClientKeyFilePath></ClientKeyFilePath> -->
        <!-- <ClientKeyPassword></ClientKeyPassword> -->
        <UseSSL>true</UseSSL>
        <Username></Username>
        <!-- <UseCBS></UseCBS> -->
        <!-- <CBSType></CBSType> -->
        <!-- <CBSAudience></CBSAudience> -->
        <!-- <UseWebsockets></UseWebsockets> -->
        <!-- <WebsocketsProtocolName></WebsocketsProtocolName> -->
        <!-- <WebsocketsRelativePath></WebsocketsRelativePath> -->
        <!-- <ManualDataSerializationFormat>StandardTagByID</ManualDataSerializationFormat> -->
        <!-- <TlsVersion>12<TlsVersion> -->
    </ManualConfigOptions>
```

Fig. 16: Empty Cloud Configuration XML File

Place the name of your IoT device in the DeviceID and ClientID fields. The Endpoint field takes the form “ssl://hostname:8883” where hostname is found in the *Overview* tab of the IoT Hub in the Azure portal. Place the SaS token obtained from Cloud Shell in the *Password* field. The Target field takes the form “devices/devicename/messages/events/” where devicename is the name of your IoT device. The *Username* field is formatted as “hostname/devicename” where hostname and devicename are as they were previously described.

```
<ManualConfigOptions>
    <DeviceID>devicename</DeviceID>
    <ClientID>devicename</ClientID>
    <Endpoint>ssl://hostname:8883</Endpoint>
    <Password>SaS token</Password>
    <Protocol>mqtt</Protocol>
    <Qos>1</Qos>
    <Target>devices/devicename/messages/events/</Target>
    <!-- <TrustStoreFilePath>\MMCMemory\Certificates\root-ca.crt</TrustStoreFilePath> -->
    <!-- <ClientCertificateFilePath>\MMCMemory\Certificates\C6CloudRouter.cert.pem</ClientCertificateFilePath> -->
    <!-- <ClientKeyFilePath>\MMCMemory\Certificates\C6CloudRouter.private.key</ClientKeyFilePath> -->
    <!-- <ClientKeyPassword></ClientKeyPassword> -->
    <UseSSL>true</UseSSL>
    <Username>hostname/devicename</Username>
    <!-- <UseCBS></UseCBS> -->
    <!-- <CBSType></CBSType> -->
    <!-- <CBSAudience></CBSAudience> -->
    <!-- <UseWebsockets></UseWebsockets> -->
    <!-- <WebsocketsProtocolName></WebsocketsProtocolName> -->
    <!-- <WebsocketsRelativePath></WebsocketsRelativePath> -->
    <!-- <ManualDataSerializationFormat>StandardTagByID</ManualDataSerializationFormat> -->
</ManualConfigOptions>
```

Fig. 17: Filled out Configuration File

During inital connection and for troubleshooting it is often helpful to enable a debug window for additional error information. To enable the debug window uncomment the tags inside of *<LoggerConfigOptions></LoggerConfigOptions>*. Save the edited configuration file.

3. Download and Start Project

Download the HMI project to the C6 Router and start the project. If the router has an internet connection, data may be pushed to Azure IoT Hub.

Test Cloud Connection

This section will review how to navigate the sample project provided with this FAQ as well as troubleshooting tips.

1. Log Data

After starting the HMI project the desktop of the router may be viewed and interacted with in Combivis Connect. Data can be logged using the LOG DATA button. Changing values of each tag can be done using the edit boxes for each respective tag.

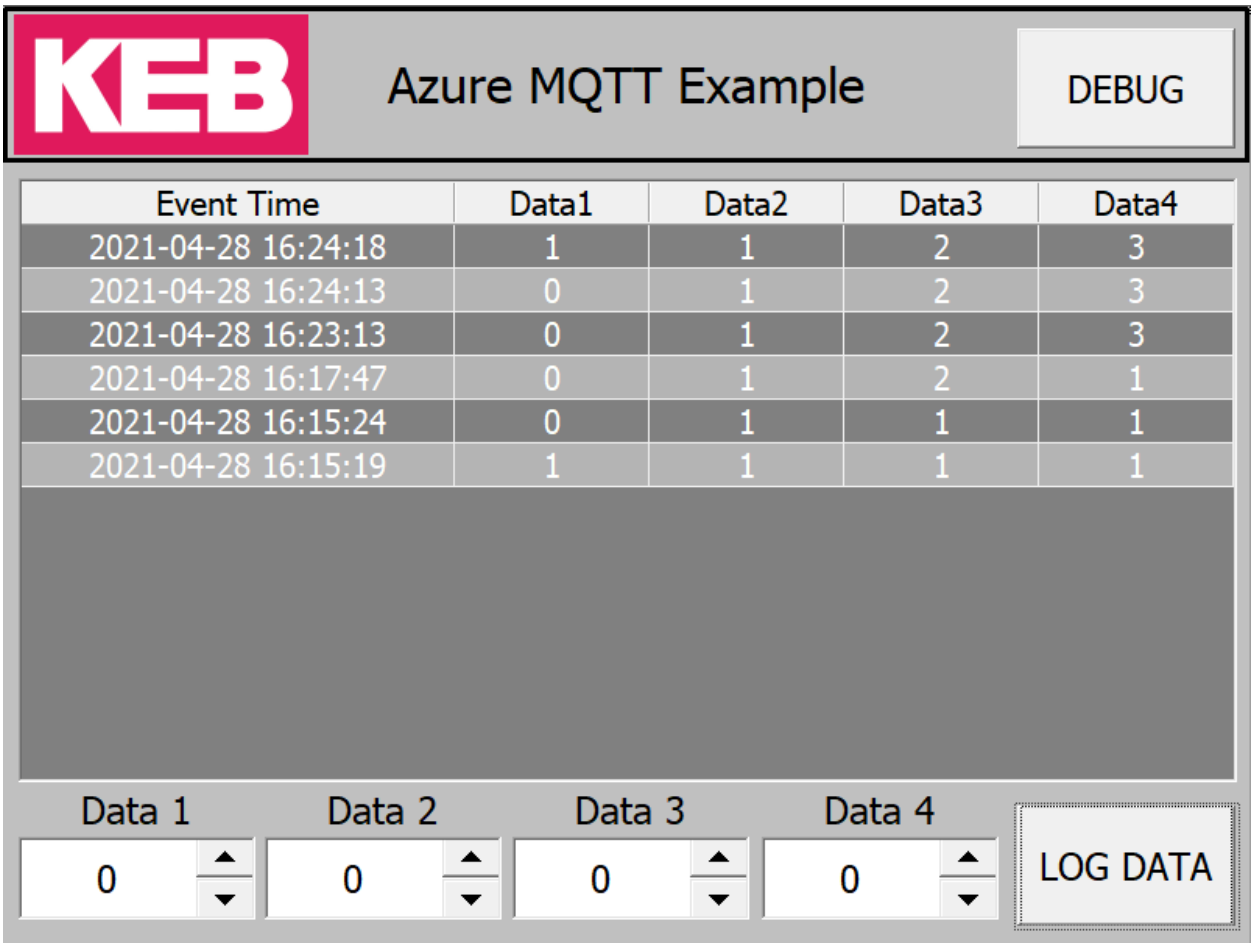

Fig. 18: Sample Project Data Screen

2. Test Debug Screen

The debug screen has been created to display the status of the connection and how many packets have been pushed since the project started.

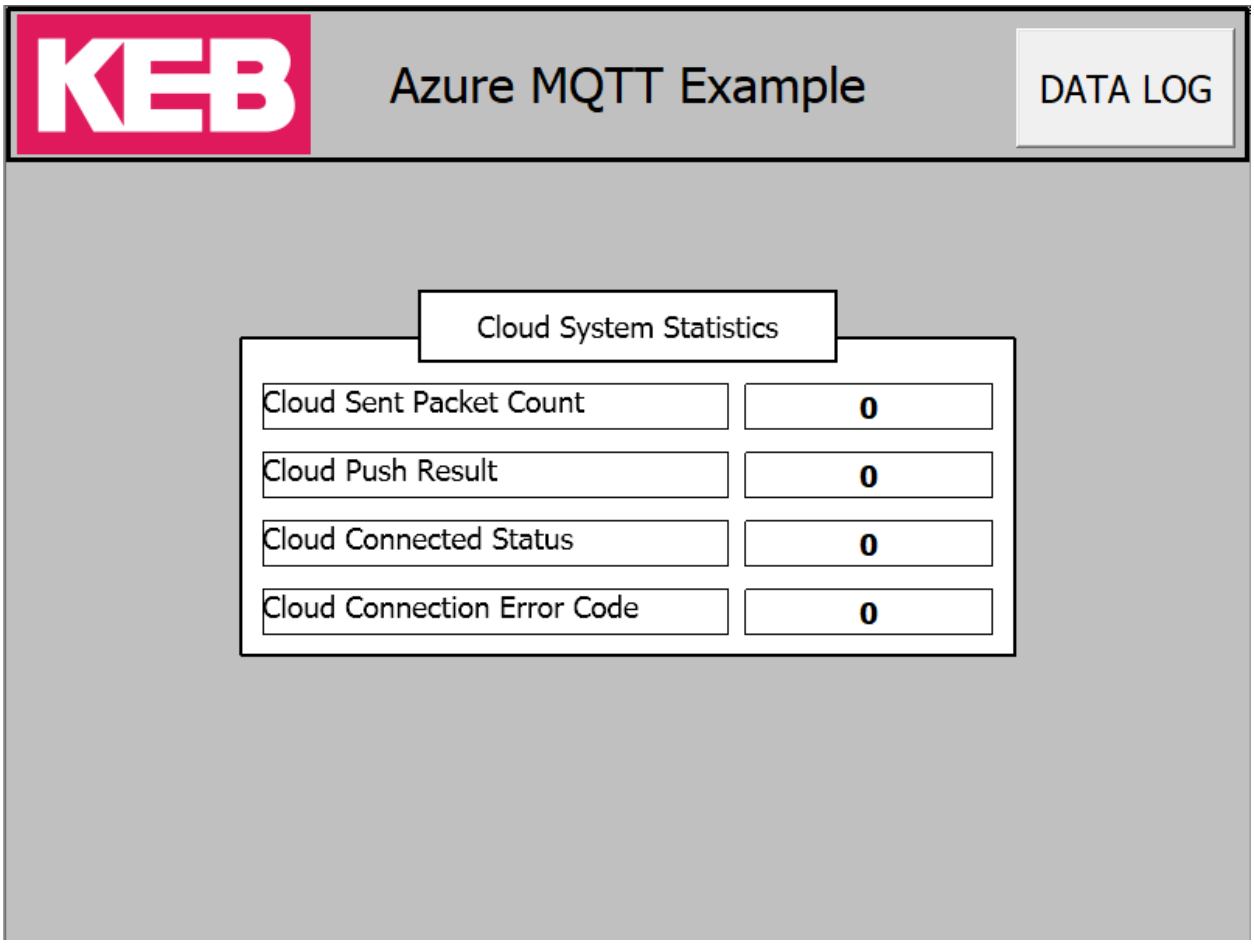

Fig. 19: Sample Project Debug Screen

3. Test Debug Window

If the debugger window has been enabled in the CloudRuntimeConfig.xml, it can be viewed by selecting the *Console* tab on the task manager of the router.

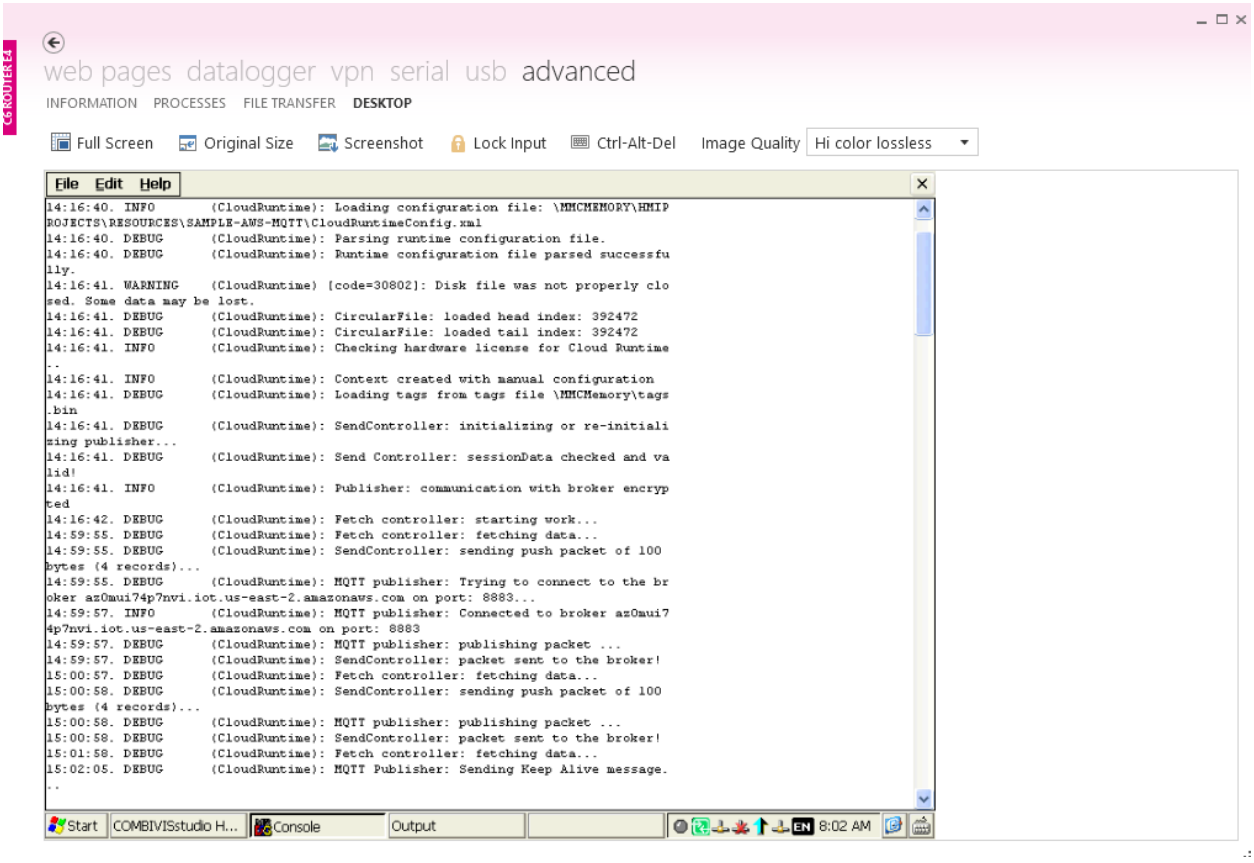

Fig. 20: Cloud Debug Window

If there are connection issues, double-check with IT that port 8883 is open for the C6 Router. This is the port used by the MQTT client.

4. Test Function App Log

There is a log in the *Code + Test* tab of the Function App where executions of the app with each push of data from the router may be viewed.

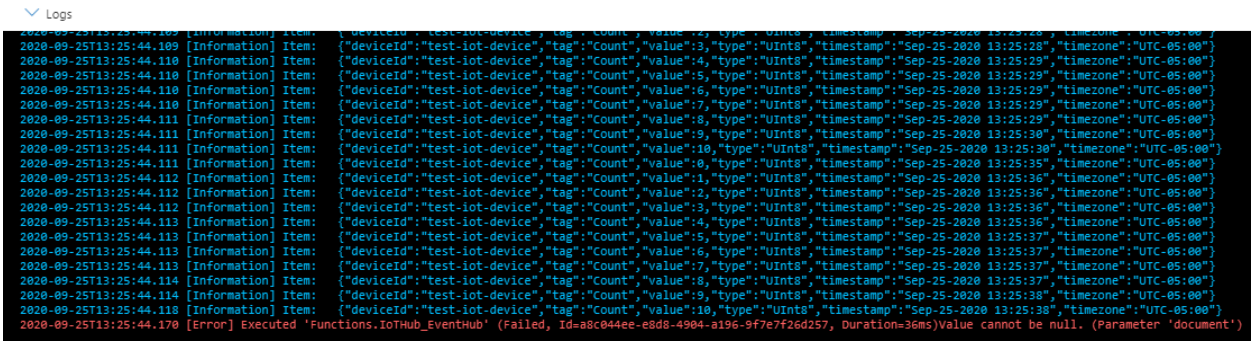

Fig. 21: Azure Function App Log Window

5. Cosmos DB Table

Pictured below is a sample output in a Cosmos DB container with the partition key “/timestamp”.

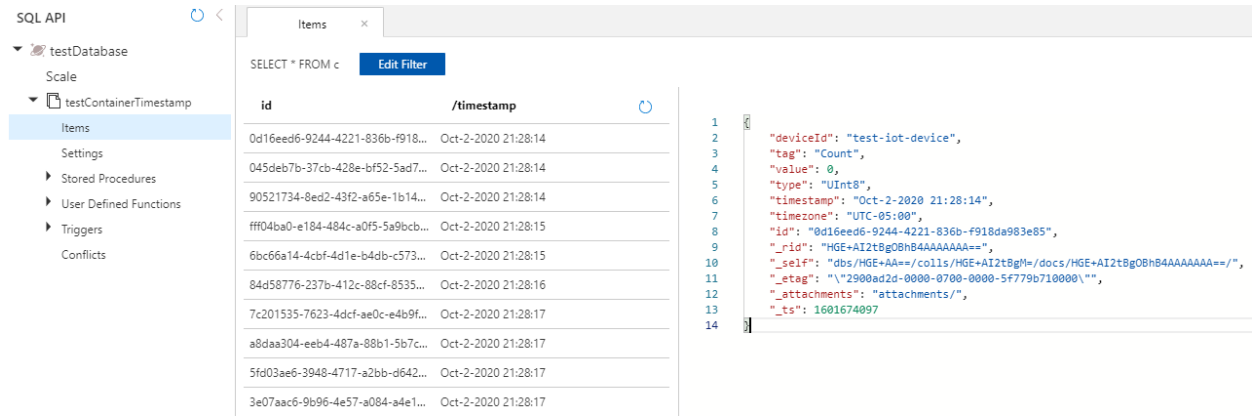

Fig. 22: Filled out Configuration File

The first five fields of deviceId, tag, value, type, and timestamp are all communicated from the router and decoded in the Function App. The fields that follow of id, _rid, etc. are assigned by Azure.

Disclaimer

KEB America, Inc. reserves the right to change/adapt specifications and technical data without prior notification. The safety and warning reference specified in this manual is not exhaustive. Although the manual and the information contained in it is made with care, KEB does not accept responsibility for misprint or other errors or resulting damages. The marks and product names are trademarks or registered trademarks of the respective title owners.

The information contained in the technical documentation, as well as any user-specific advice in verbal or in written form are made to the best of our knowledge and information about the application. However, they are considered for information only without responsibility. This also applies to any violation of industrial property rights of a third-party.

Inspection of our units in view of their suitability for the intended use must be done generally by the user. Inspections are particular necessary, if changes are executed, which serve for the further development or adaption of our products to the applications (hardware, software or download lists). Inspections must be repeated completely, even if only parts of hardware, software or download lists are modified.

Application and use of our units in the target products is outside of our control and therefore lies exclusively in the area of responsibility of the user.

Americas:

KEB America, Inc.
5100 Valley Industrial Blvd South
Shakopee, MN 55379, USA
(+1) 952-224-1400
info@kebamerica.com

Headquarters:

KEB Automation KG
Suedstrasse 38
D - 32683 Barntrup, Germany
(+49) 5263 401-0
info@keb.de