Connecting KD240 to AZURE IoT

2023.12.26/v0.1 Sanam Shakya

Creating Petalinux Project

Create KD240 Petalinux Project from BSP

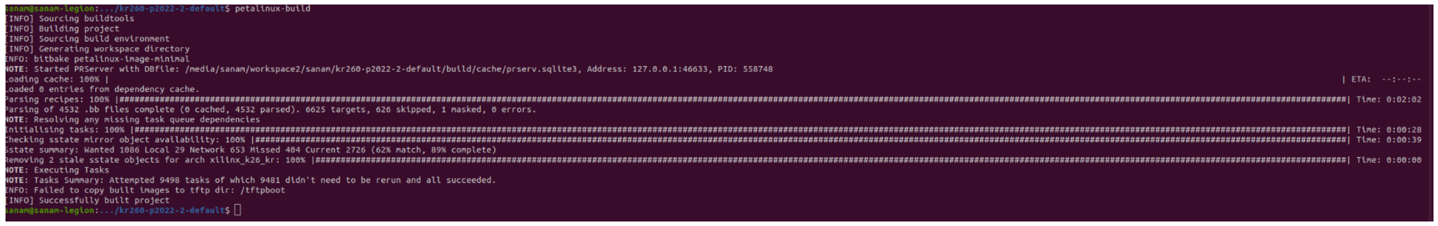
In host machine, create the petalinux project using BSP available at following download link:
https://www.xilinx.com/member/forms/download/xef.html?filename=kria-image-full-cmdline-k26-smk-20230913102327.rootfs.wic.xz
Run following commands to create petalinux project after sourcing the Petalinux 2023.1 environment:

```
$ source <Path to Petalinux 2023.1>/settings.sh
$ petalinux-create -t project -s xilinx-kd240-starterkit-v2023.1-09130455.bsp -n
kd240-2023_1-default
$ cd kd240-2023_1-default
```

Here “kd240-2023_1-default” is the directory created by petalinux-create command, you can change the name according to your need. This directory is the petalinux-project base directory, which we will be using in further steps.
Next build the project:

```
$ petalinux-config --silentconfig
$ petalinux-build
```

Here is the console log after running the petalinux-build.



```
sanam@sanam-legion:.../kr260-p2022-2-default$ petalinux-build
[INFO] Sourcing buildtools
[INFO] Building project
[INFO] Sourcing build environment
[INFO] Generating workspace directory
INFO: bitbake petalinux-image-minimal
NOTE: Started PRServer with DBfile: /media/sanam/workspace2/sanam/kr260-p2022-2-default/build/cache/prserv.sqlite3, Address: 127.0.0.1:46633, PID: 558748
Loading cache: 100% |                                                                                                   | ETA:  --:--:--
Loaded 0 entries from dependency cache.
Parsing recipes: 100% |##############################################################################################| Time: 0:02:02
Parsing of 4532 .bb files complete (0 cached, 4532 parsed). 6625 targets, 626 skipped, 1 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% |###########################################################################################| Time: 0:00:28
Checking sstate mirror object availability: 100% |###################################################################| Time: 0:00:39
Sstate summary: Wanted 1086 Local 29 Network 653 Missed 404 Current 2726 (62% match, 89% complete)
Removing 2 stale sstate objects for arch xilinx_k26_kr: 100% |#######################################################| Time: 0:00:00
NOTE: Executing Tasks
NOTE: Tasks Summary: Attempted 9498 tasks of which 9481 didn't need to be rerun and all succeeded.
INFO: Failed to copy built images to tftp dir: /tftpboot
[INFO] Successfully built project
sanam@sanam-legion:.../kr260-p2022-2-default$
```

Next create the SD card image with the following commands:

```
petalinux-package --wic --images-dir images/linux/ --bootfiles
"ramdisk.cpio.gz.u-boot,boot.scr,Image,system.dtb,system-zynqmp-sck-kd-g-revA.dtb"
--disk-name "sda"
```

This will create the petalinux-sdimage.wic image at <petalinux project directory>/image/linux folder. Copy the created wic image to SD card using tools like Balena Etcher.

Installing hardware overlay

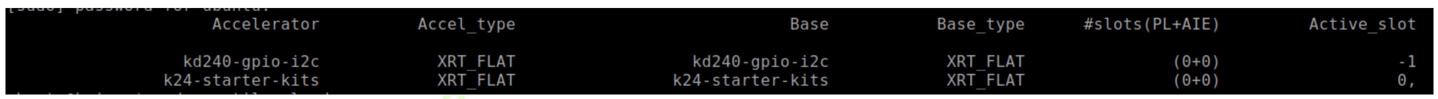
Get the KD240 firmware folder. It contains:

- kd240-gpio-i2c.bit.bin
- kd240-gpio-i2c.dtbo
- shell.json

Copy these file to the KD240 board. For firmware to be loaded using xmutil (FPGA manager), one has to copy these file at “/lib/firmware/xilinx”.
For this create the folder at “kd240-gpio-i2c” at “/lib/firmware/xilinx” and copy the files in “kd240-gpio-i2c” folder.

```
cd /lib/firmware/xilinx
sudo mkdir kd240-gpio-i2c
sudo cp <kd240-firmware directory>/krc260_i2c* ./
sudo cp <kd240-firmware directory>/shell.json ./
```

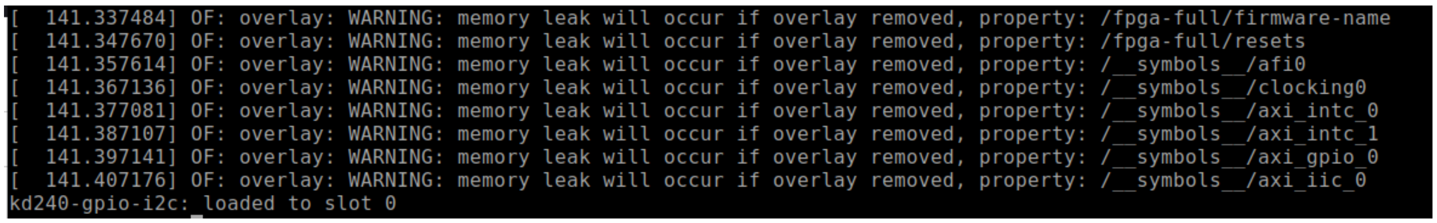
Next, check the available fpga firmware using `xmutil listapps` command. `kd240-i2c` will be available in the list.



```
            Accelerator          Accel_type                      Base        Base_type      #slots(PL+AIE)       Active_slot

         kd240-gpio-i2c            XRT_FLAT            kd240-gpio-i2c         XRT_FLAT               (0+0)                -1
      k24-starter-kits             XRT_FLAT          k24-starter-kits         XRT_FLAT               (0+0)                0,
```

Next load the `kd240-gpio-i2c` firmware, which contains necessary hardwares(gpio) and interfaces. In our Greengrass Demo we will be using these gpio to trigger the publishing data to AWS Greengrass IoT cloud server and also actuate GPIO on the message received from AWS cloud.

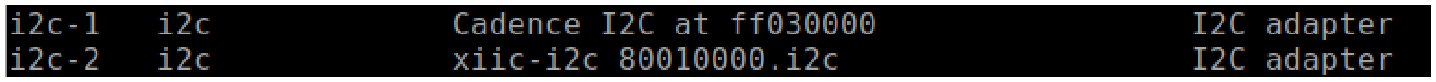
```
sudo xmutil unloadapp
sudo xmutil loadapp kd240-gpio-i2c
```



```
[  141.337484] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /fpga-full/firmware-name
[  141.347670] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /fpga-full/resets
[  141.357614] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/afi0
[  141.367136] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/clocking0
[  141.377081] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/axi_intc_0
[  141.387107] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/axi_intc_1
[  141.397141] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/axi_gpio_0
[  141.407176] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/axi_iic_0
kd240-gpio-i2c: loaded to slot 0
```

Now, check the available i2c channels available in the system using `i2cdetect` i2c utility tool.

```
sudo i2cdetect -l
```



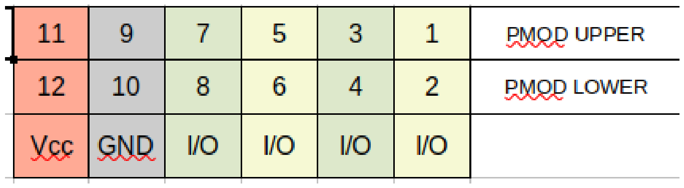
```
i2c-1   i2c          Cadence I2C at ff030000            I2C adapter
i2c-2   i2c          xiic-i2c 80010000.i2c              I2C adapter
```

`i2c-2` channel will be used to connect to BMP180 sensor.

Connecting BMP180 to AXI I2C Bus

Connect BMP180 sensors, Vcc, GND, I2C SDA and I2C SCLK pins to PMOD as explained below:
PMOD1-> 3 - I2C SCLK

PMOD1-> 1 - I2C SDA
PMOD1-> GND - BMP180 GND
PMOD1->Vcc - BMP180 Vcc

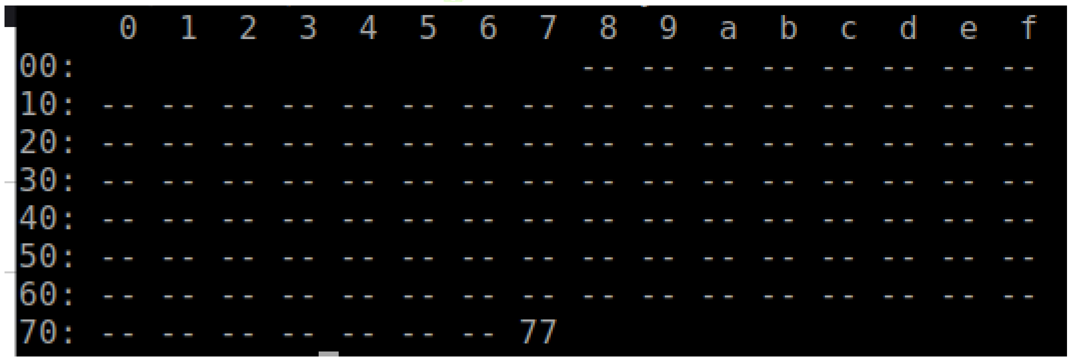


11	9	7	5	3	1	PMOD UPPER
12	10	8	6	4	2	PMOD LOWER
Vcc	GND	I/O	I/O	I/O	I/O	

PMOD port numbering

After connecting BMP180 sensor to KD240 PMOD port, use i2c utility tools to scan for the available devices in i2c-8 channel.

```
sudo i2cdetect -y 2
```

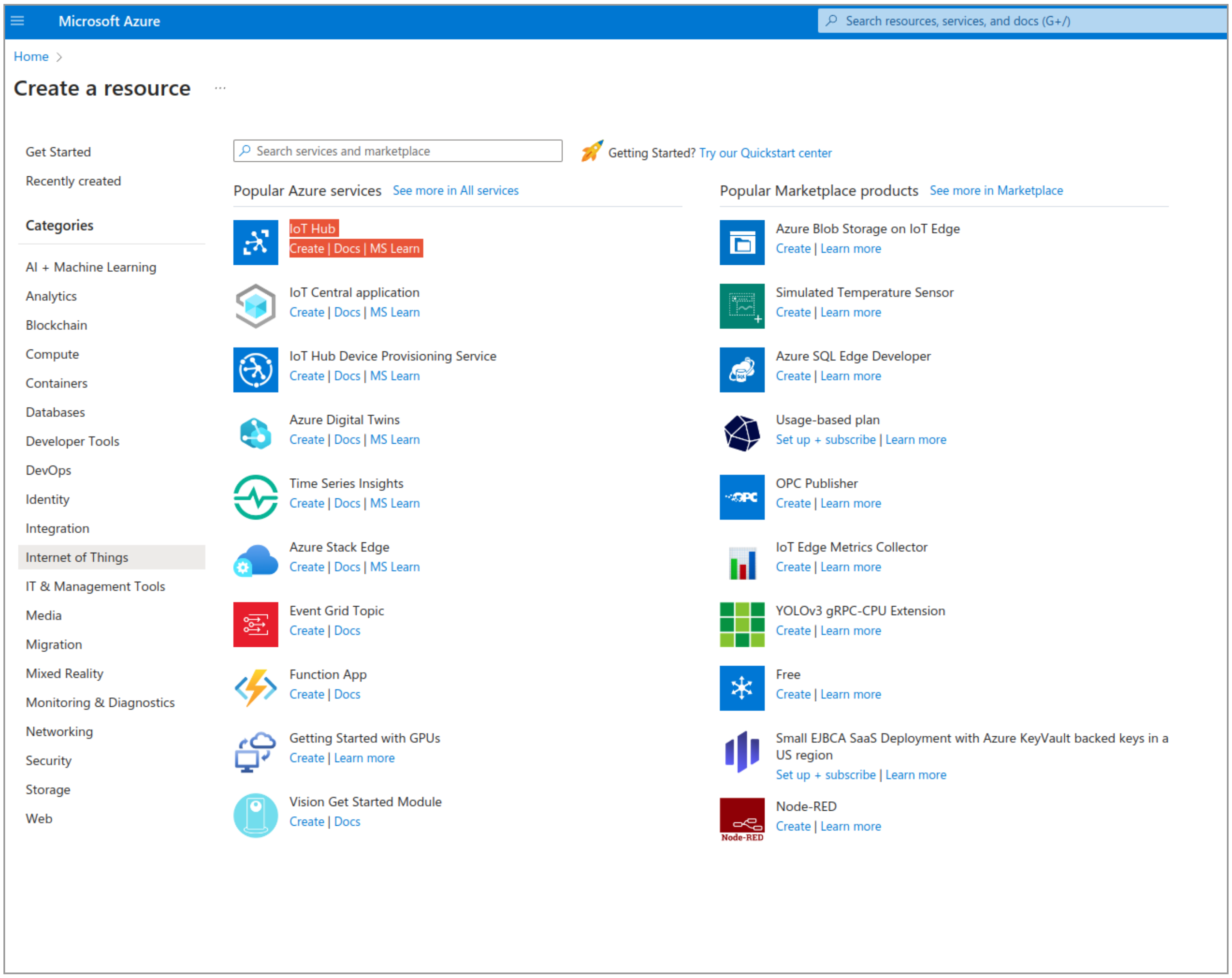


```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- 77
```

In i2c scan, we find a device is available at address ‘77’, which corresponds to BMP180 i2c sensor.
Next we will add the component for publishing BMP180 sensor data to the AWS IoT cloud.

Create IoT Hub in Azure Portal:

- Go to Azure portal " https://portal.azure.com ".
- Create a resource >> IoT Hub.


Microsoft Azure | Search resources, services, and docs (G+/)

Home >

Create a resource ···

Get Started
Recently created

Categories

AI + Machine Learning
Analytics
Blockchain
Compute
Containers
Databases
Developer Tools
DevOps
Identity
Integration
Internet of Things
IT & Management Tools
Media
Migration
Mixed Reality
Monitoring & Diagnostics
Networking
Security
Storage
Web

Search services and marketplace

Getting Started? Try our Quickstart center

Popular Azure services — See more in All services

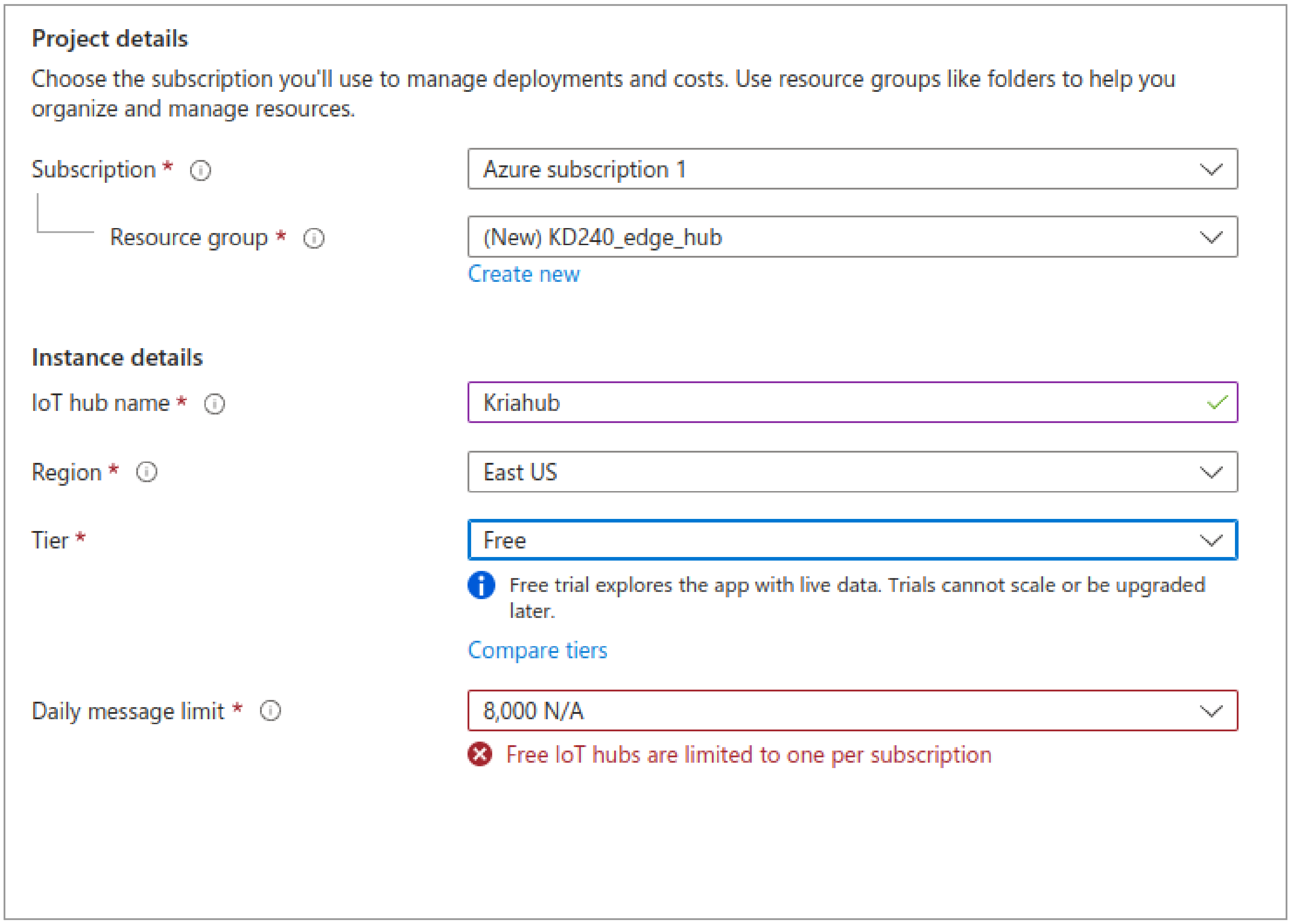
- IoT Hub — Create | Docs | MS Learn
- IoT Central application — Create | Docs | MS Learn
- IoT Hub Device Provisioning Service — Create | Docs | MS Learn
- Azure Digital Twins — Create | Docs | MS Learn
- Time Series Insights — Create | Docs | MS Learn
- Azure Stack Edge — Create | Docs | MS Learn
- Event Grid Topic — Create | Docs
- Function App — Create | Docs
- Getting Started with GPUs — Create | Learn more
- Vision Get Started Module — Create | Docs

Popular Marketplace products — See more in Marketplace

- Azure Blob Storage on IoT Edge — Create | Learn more
- Simulated Temperature Sensor — Create | Learn more
- Azure SQL Edge Developer — Create | Learn more
- Usage-based plan — Set up + subscribe | Learn more
- OPC Publisher — Create | Learn more
- IoT Edge Metrics Collector — Create | Learn more
- YOLOv3 gRPC-CPU Extension — Create | Learn more
- Free — Create | Learn more
- Small EJBCA SaaS Deployment with Azure KeyVault backed keys in a US region — Set up + subscribe | Learn more
- Node-RED — Create | Learn more

Next, create one IoT Hub Service and fill in the necessary details

Next, create one IoT Hub Service and fill in the necessary details



Project details

Choose the subscription you'll use to manage deployments and costs. Use resource groups like folders to help you organize and manage resources.

Subscription *	Azure subscription 1
Resource group *	(New) KD240_edge_hub

Instance details

IoT hub name *	Kriahub
Region *	East US
Tier *	Free

Free trial explores the app with live data. Trials cannot scale or be upgraded later.

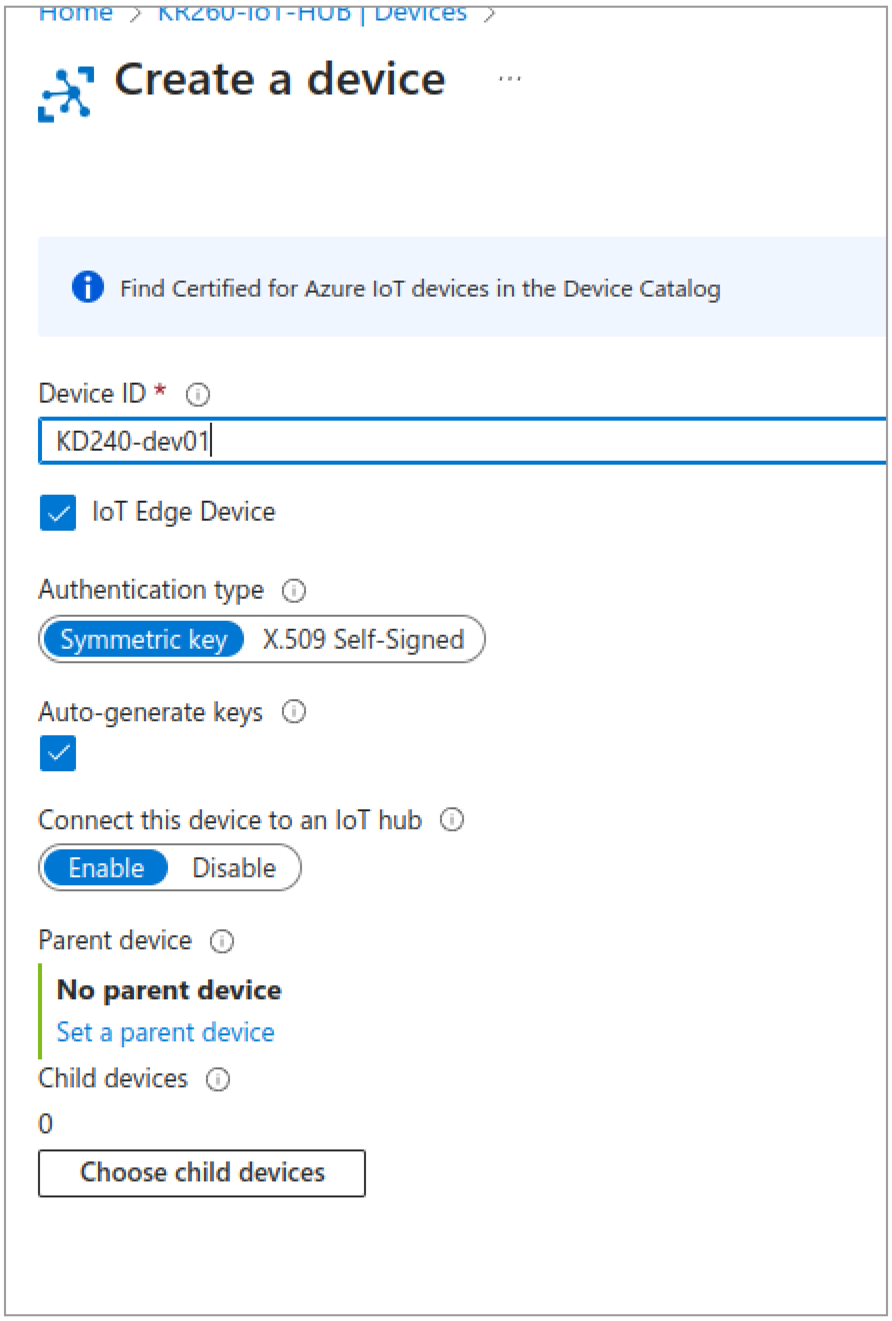
Daily message limit *	8,000 N/A

Free IoT hubs are limited to one per subscription

- Click on Review+ Create button to create the Azure IoT Hub.
- Next, create a device where you can actually receive some data from the hardware.

Create an IoT Device

Go to the IoT Device and click on new, and give the device ID


Home > KR260-IoT-HUB | Devices >

Create a device

Find Certified for Azure IoT devices in the Device Catalog

Device ID *: KD240-dev01

✓ IoT Edge Device

Authentication type: [✓ Symmetric key] X.509 Self-Signed

Auto-generate keys: ✓

Connect this device to an IoT hub: [✓ Enable] Disable

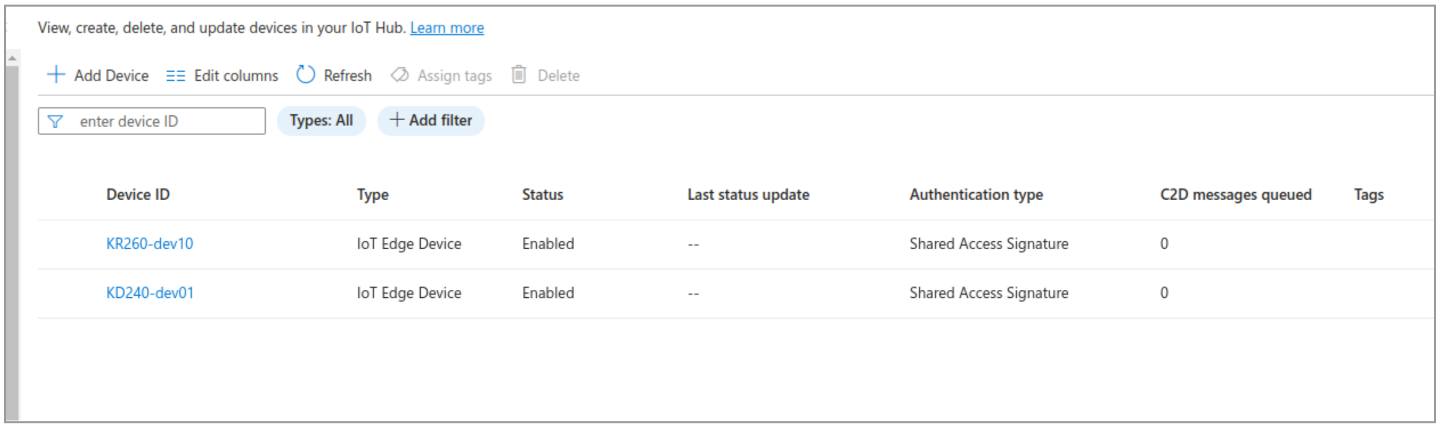
Parent device: **No parent device**

Set a parent device

Child devices: 0

Choose child devices

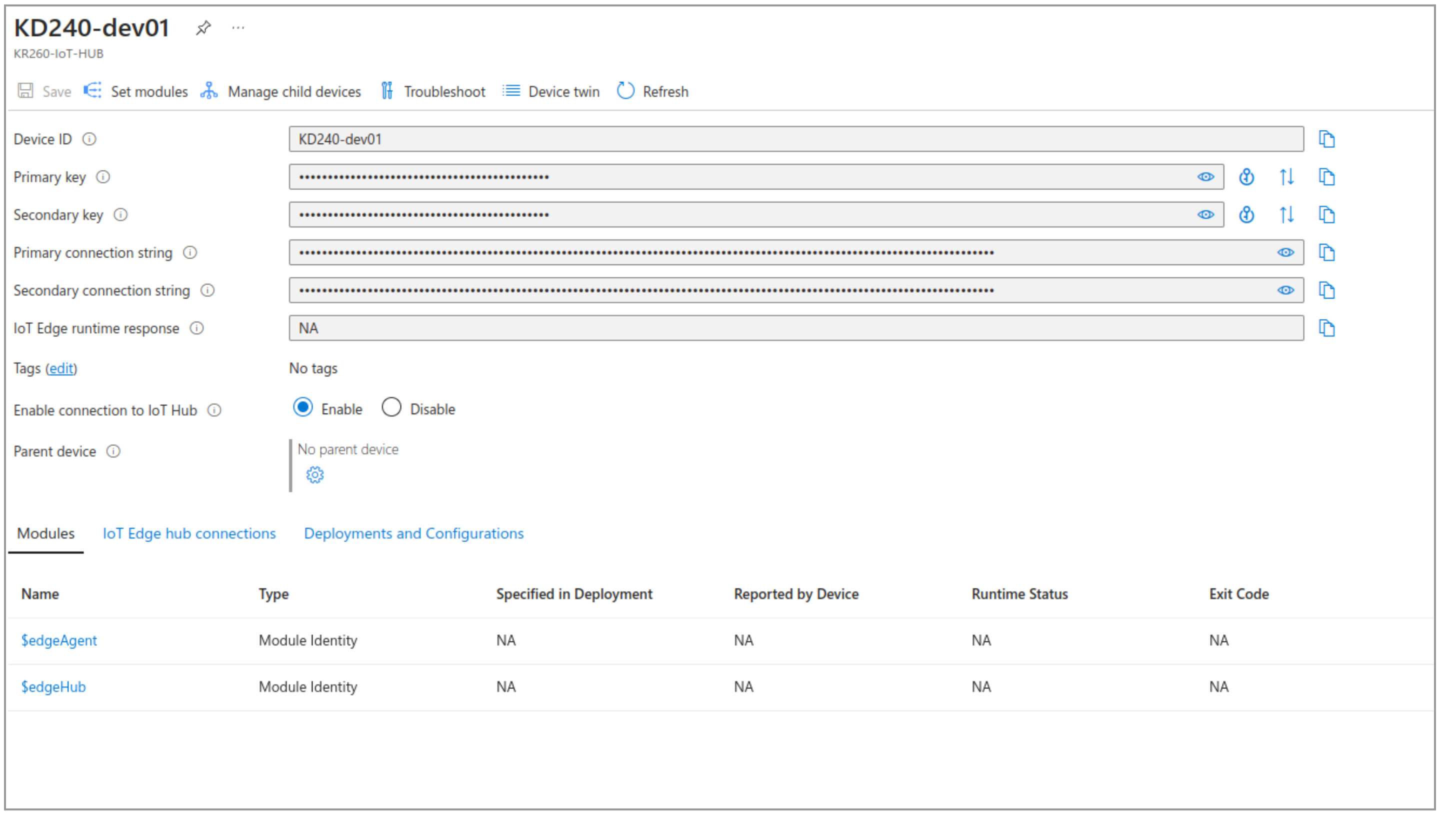
After this device will be available in the IoT hub Device list.



View, create, delete, and update devices in your IoT Hub. Learn more

Device ID	Type	Status	Last status update	Authentication type	C2D messages queued	Tags
KR260-dev10	IoT Edge Device	Enabled	--	Shared Access Signature	0	
KD240-dev01	IoT Edge Device	Enabled	--	Shared Access Signature	0	

Next, look into device information for getting the keys and connection string.



KD240-dev01
KR260-IoT-HUB

Device ID	KD240-dev01
Primary key	••
Secondary key	••
Primary connection string	•••
Secondary connection string	•••
IoT Edge runtime response	NA
Tags (edit)	No tags
Enable connection to IoT Hub	[✓ Enable] Disable
Parent device	No parent device

Modules | IoT Edge hub connections | Deployments and Configurations

Name	Type	Specified in Deployment	Reported by Device	Runtime Status	Exit Code
$edgeAgent	Module Identity	NA	NA	NA	NA
$edgeHub	Module Identity	NA	NA	NA	NA

Copy the "Primary Connection String" which will be used in the python application for sending the sensor data to IoT hub.

Installing python packages

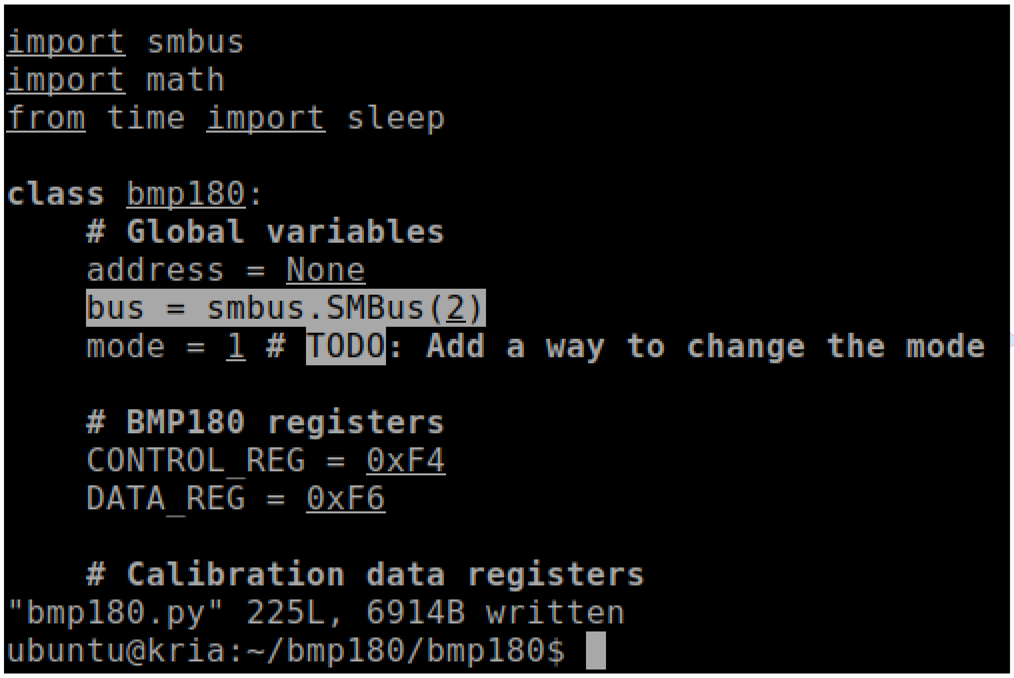
azure.iot.device python module is required to create a azure IoT device at the edge device. Install it using python pip3:

```
sudo pip3 install azure-iot-device
```

Further for getting 'bmp180' sensor data from i2c, install bmp180 python driver module from git. For installing, run following commands:

```
git clone https://github.com/m-rtijn/bmp180
cd bmp180
```

Update the ~/bmp180/bmp180/bmp180.py to use i2c-2 channel by changing following lines:



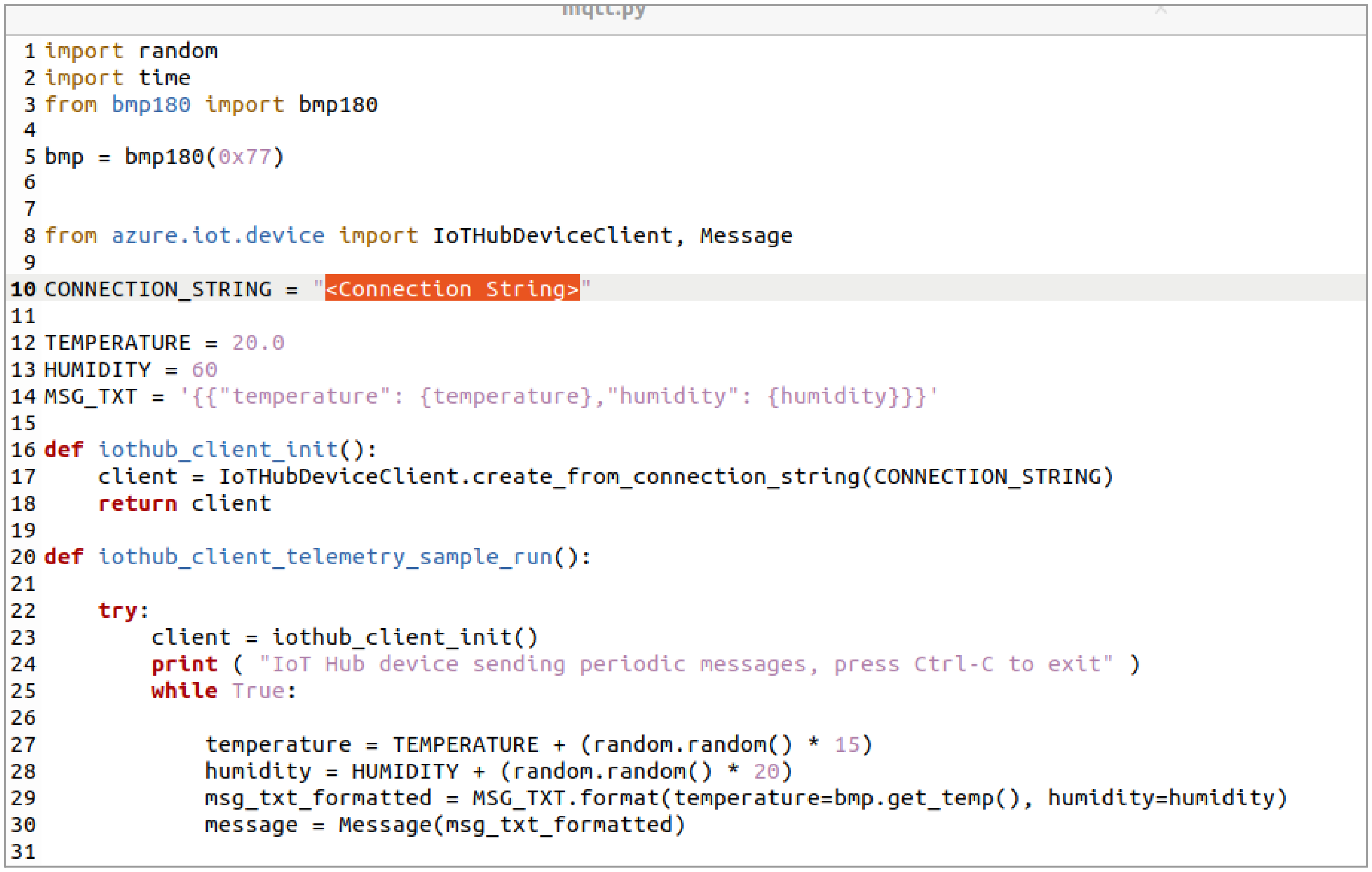
```
import smbus
import math
from time import sleep

class bmp180:
    # Global variables
    address = None
    bus = smbus.SMBus(2)
    mode = 1 # TODO: Add a way to change the mode

    # BMP180 registers
    CONTROL_REG = 0xF4
    DATA_REG = 0xF6

    # Calibration data registers
"bmp180.py" 225L, 6914B written
ubuntu@kria:~/bmp180/bmp180$
```

Install the bmp180 module by running:

```
sudo python3 setup.py install
```

Adding python application in KRIA

Copy the azure_bmp180.py example code to the KD240 board.
Next update the “CONNECTION STRING” with the above Primary Connection string.

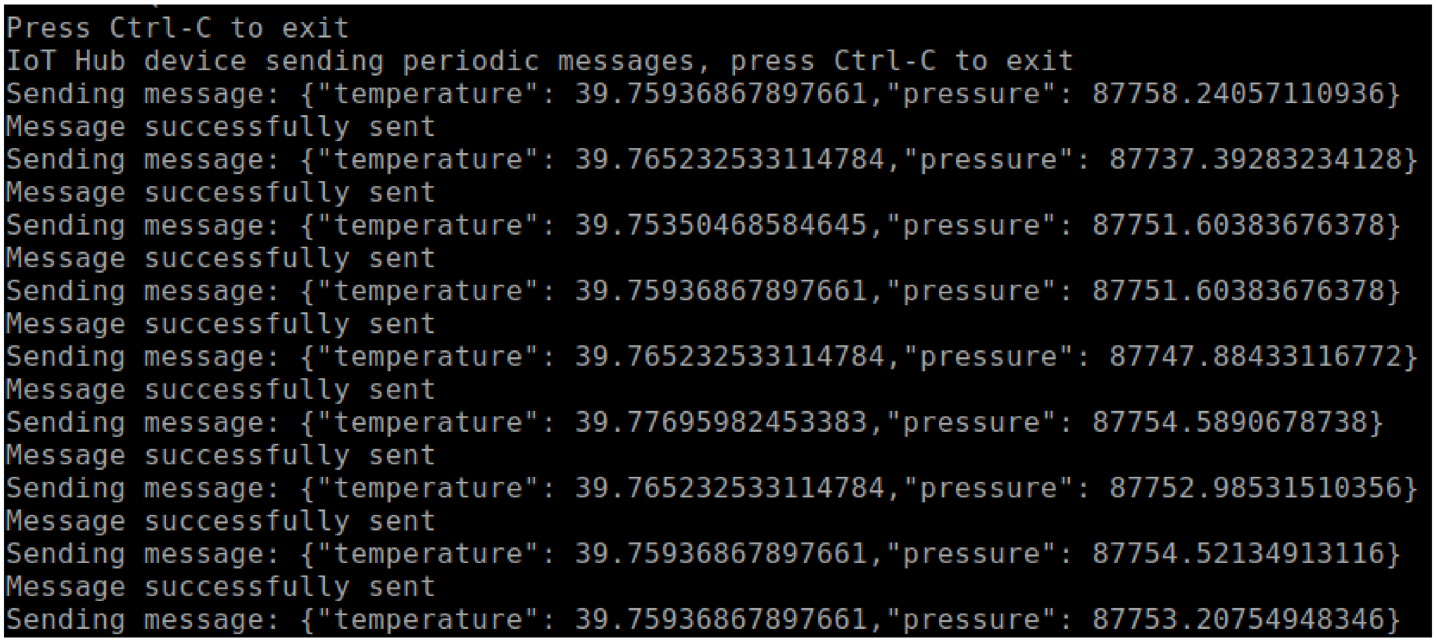


```python
import random
import time
from bmp180 import bmp180

bmp = bmp180(0x77)


from azure.iot.device import IoTHubDeviceClient, Message

CONNECTION_STRING = "<Connection String>"

TEMPERATURE = 20.0
HUMIDITY = 60
MSG_TXT = '{{"temperature": {temperature},"humidity": {humidity}}}'

def iothub_client_init():
    client = IoTHubDeviceClient.create_from_connection_string(CONNECTION_STRING)
    return client

def iothub_client_telemetry_sample_run():

    try:
        client = iothub_client_init()
        print ( "IoT Hub device sending periodic messages, press Ctrl-C to exit" )
        while True:

            temperature = TEMPERATURE + (random.random() * 15)
            humidity = HUMIDITY + (random.random() * 20)
            msg_txt_formatted = MSG_TXT.format(temperature=bmp.get_temp(), humidity=humidity)
            message = Message(msg_txt_formatted)

```

Then run the application in console:

```
sudo python3 azure_bmp180.py
```

Here is the console log after a successful message send to Azure IoT hub.

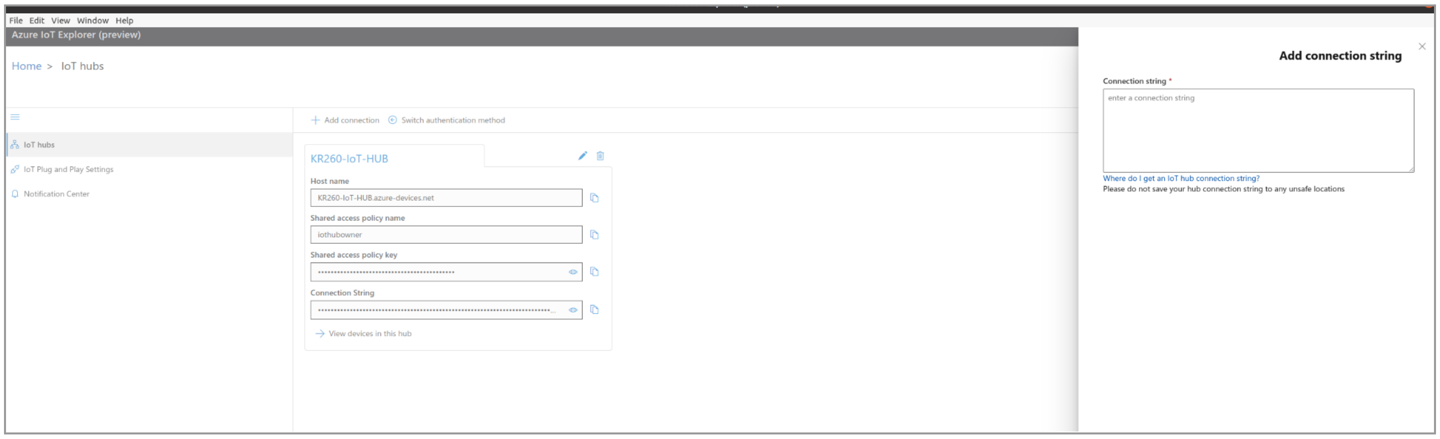


```
Press Ctrl-C to exit
IoT Hub device sending periodic messages, press Ctrl-C to exit
Sending message: {"temperature": 39.75936867897661,"pressure": 87758.24057110936}
Message successfully sent
Sending message: {"temperature": 39.765232533114784,"pressure": 87737.39283234128}
Message successfully sent
Sending message: {"temperature": 39.75350468584645,"pressure": 87751.60383676378}
Message successfully sent
Sending message: {"temperature": 39.75936867897661,"pressure": 87751.60383676378}
Message successfully sent
Sending message: {"temperature": 39.765232533114784,"pressure": 87747.88433116772}
Message successfully sent
Sending message: {"temperature": 39.77695982453383,"pressure": 87754.5890678738}
Message successfully sent
Sending message: {"temperature": 39.765232533114784,"pressure": 87752.98531510356}
Message successfully sent
Sending message: {"temperature": 39.75936867897661,"pressure": 87754.52134913116}
Message successfully sent
Sending message: {"temperature": 39.75936867897661,"pressure": 87753.20754948346}
```

Viewing message in Host Machine

For viewing the message published by Azure IoT Device in KD240, one can use Azure IoT explorer available in following link:
https://github.com/Azure/azure-iot-explorer/releases

In IoT HUbs page of the application, in +Add connection copy the connection string for the IoT hub and save the configs:


File Edit View Window Help
Azure IoT Explorer (preview)
Home > IoT hubs
IoT hubs
IoT Plug and Play Settings
Notification Center
+ Add connection
Switch authentication method
View devices in this hub
Add connection string
enter a connection string

KR260-IoT-HUB

Host name
KR260-IoT-HUB.azure-devices.net

Shared access policy name
iothubowner

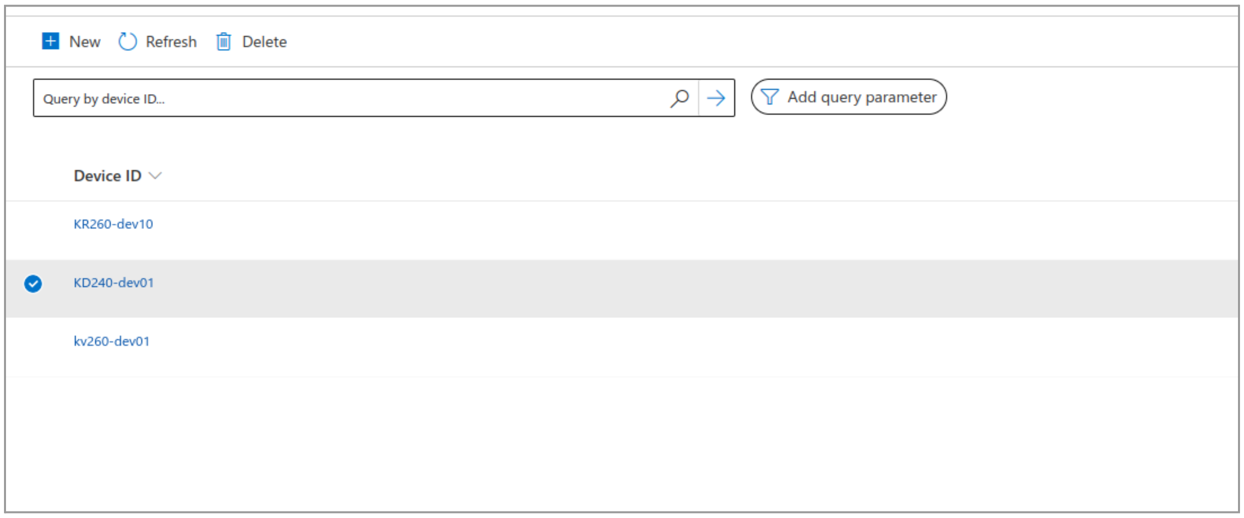
Shared access policy key
••

Connection String
••...

Connection string *

Where do I get an IoT hub connection string?
Please do not save your hub connection string to any unsafe locations

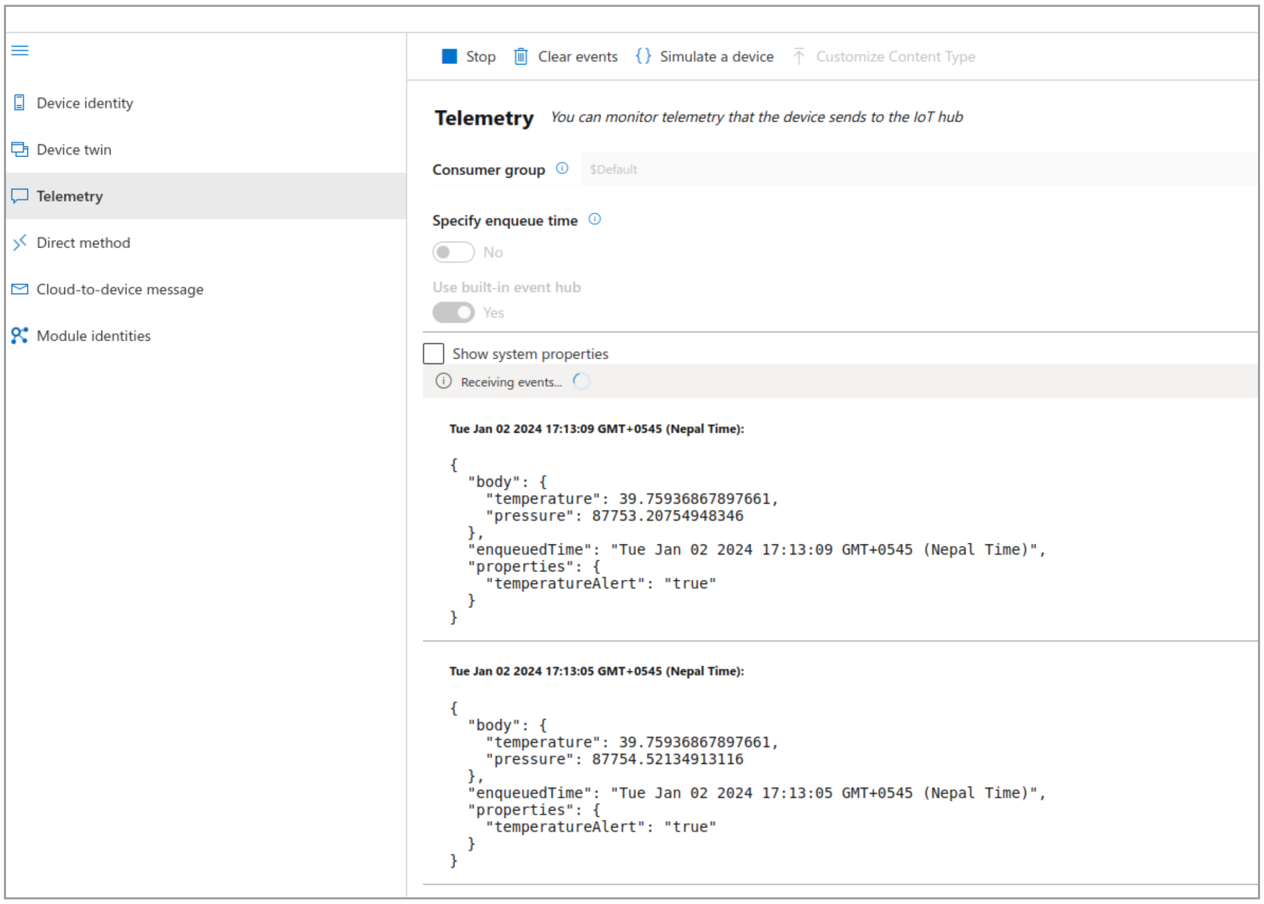
One can find the corresponding device list in the IoT HuB page of Azure IoT explorer application.



KR260-dev10
KD240-dev01
kv260-dev01

Just click onto the device to view the device information and also the message send by python application running in the KD240 board.

For viewing the message send to device, go to Telemetry and click the >Start button.
After this one can view the message send to the device.



Consumer group: $Default

Tue Jan 02 2024 17:13:09 GMT+0545 (Nepal Time):

```
{
  "body": {
    "temperature": 39.75936867897661,
    "pressure": 87753.20754948346
  },
  "enqueuedTime": "Tue Jan 02 2024 17:13:09 GMT+0545 (Nepal Time)",
  "properties": {
    "temperatureAlert": "true"
  }
}
```

Tue Jan 02 2024 17:13:05 GMT+0545 (Nepal Time):

```
{
  "body": {
    "temperature": 39.75936867897661,
    "pressure": 87754.52134913116
  },
  "enqueuedTime": "Tue Jan 02 2024 17:13:05 GMT+0545 (Nepal Time)",
  "properties": {
    "temperatureAlert": "true"
  }
}
```