Lab 2: Raspberry Pi, Python, Azure IoT Central, and Docker Container Debugging

Author	**Dave Glover, Microsoft Cloud Developer Advocate**
Platforms	Linux, macOS, Windows, Raspbian Buster
Services	Azure IoT Central
Tools	Visual Studio Code Insiders Edition
Language	Python
Date	As of August, 2019

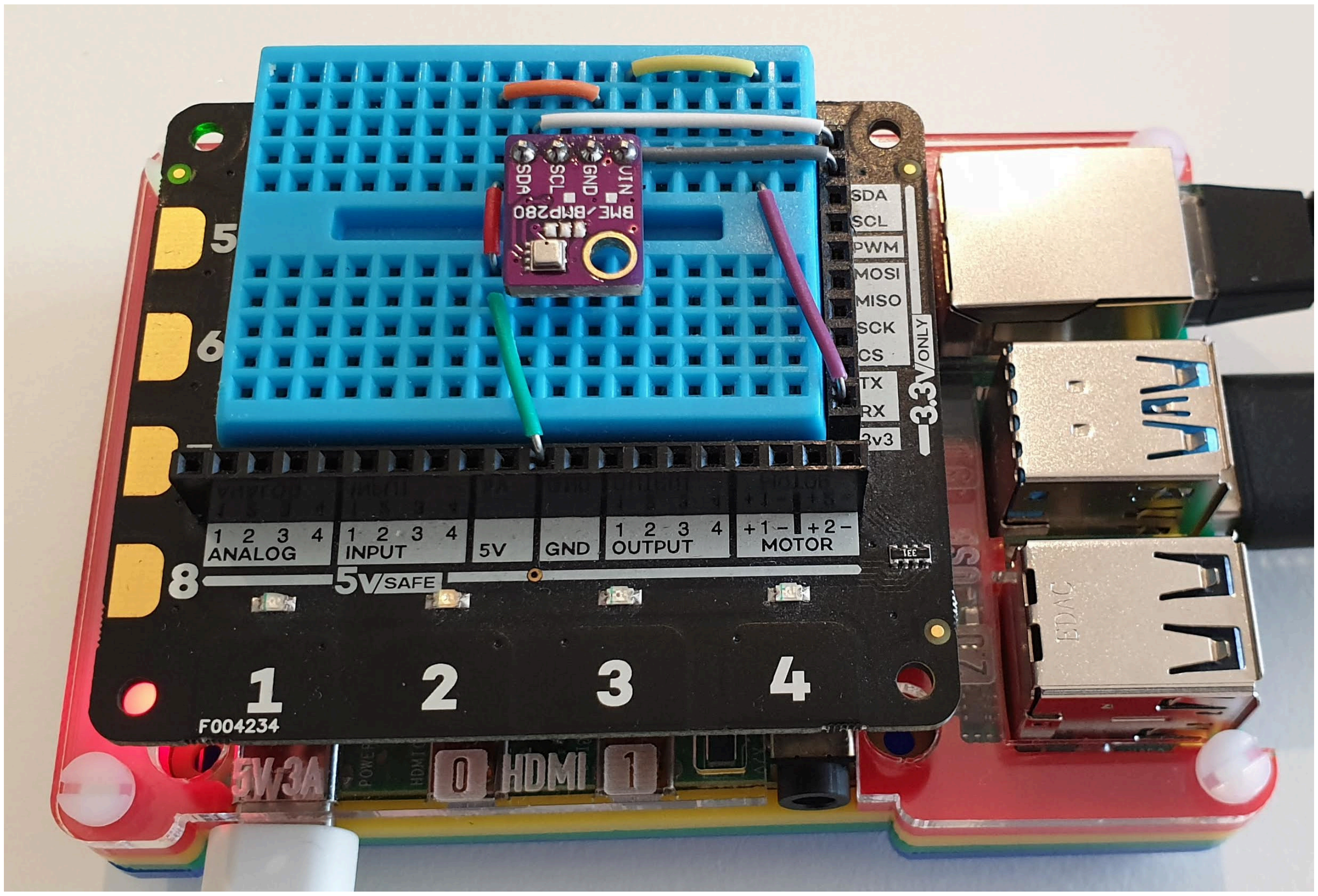
Follow me on Twitter @dglover

PDF Lab Guide

You may find it easier to download and follow the PDF version of the Raspberry Pi, Python, Azure IoT Central, and Docker Container Debugging hands-on lab guide.

Introduction

In this hands-on lab, you will learn how to create an Internet of Things (IoT) Python application with Visual Studio Code, run it in a Docker Container on a Raspberry Pi, read the temperature, humidity, and air pressure telemetry from a BME280 sensor, then attach, and debug the Python code running in the container.



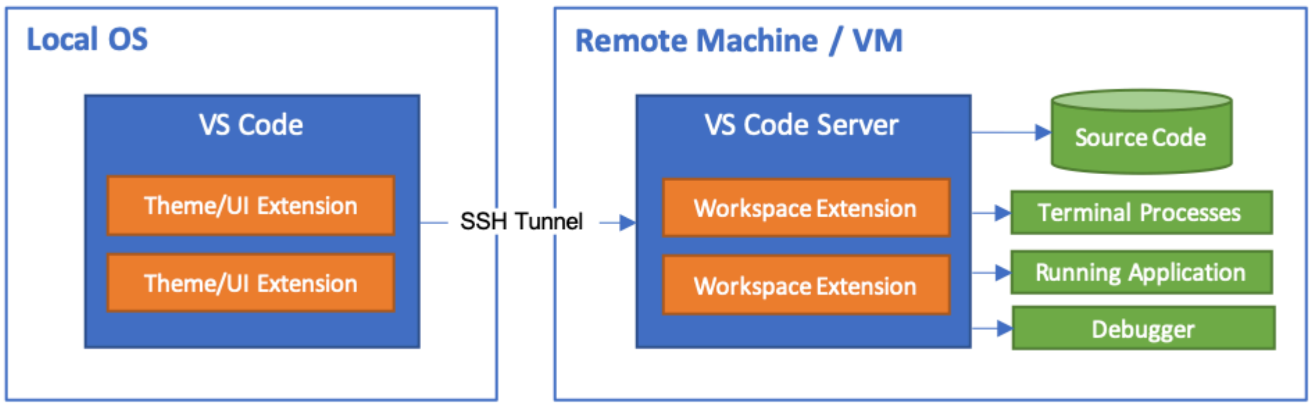
References

- Visual Studio Code
- Azure IoT Central
- Installing Docker on Raspberry Pi Buster
- Understanding Docker in 12 Minutes

Remote Development using SSH

The Visual Studio Code Remote - SSH extension allows you to open a remote folder on any remote machine, virtual machine, or container with a running SSH server and take full advantage of Visual Studio Code's feature set. Once connected to a server, you can interact with files and folders anywhere on the remote filesystem.

No source code needs to be on your local machine to gain these benefits since the extension runs commands and other extensions directly on the remote machine.



CircuitPython

CircuitPython, an Adafruit initiative, is built on the amazing work of Damien George and the MicroPython community. CircuitPython adds hardware support for Microcontroller development and simplifies some aspects of MicroPython. MicroPython and by extension, CircuitPython implements version 3 of the Python language reference. So, your Python 3 skills are transferrable.

CircuitPython runs on over 60 **Microcontrollers** as well as the **Raspberry Pi**. This means you build applications that access GPIO hardware on a Raspberry Pi using CircuitPython libraries.

The advantage of running CircuitPython on the Raspberry Pi is that there are powerful Python debugging tools available. If you have ever tried debugging applications on a Microcontroller, then you will appreciate it can be painfully complex and slow. You resort to print statements, toggling the state of LEDs, and worst case, using specialized hardware.

With Raspberry Pi and CircuitPython, you build and debug on the Raspberry Pi, when it is all working you transfer the app to a CircuitPython Microcontroller. You need to ensure any libraries used are copied to the Microcontroller, and pin mappings are correct. But much much simpler!

This hands-on lab uses CircuitPython libraries for GPIO, I2C, and the BME280 Temperature/Pressure/Humidity sensor. The CircuitPython libraries are installed on the Raspberry Pi with pip3.

```
pip3 install adafruit-blinka adafruit-circuitpython-bme280
```

Software Installation

This hands-on lab uses Visual Studio Code. Visual Studio Code is a code editor and is one of the most popular **Open Source** projects on GitHub. It runs on Linux, macOS, and Windows.

Install:

1. Visual Studio Code Insiders Edition
   As at August 2019, **Visual Studio Code Insiders Edition** is required as it has early support for Raspberry Pi and Remote Development over SSH.
2. Remote - SSH Visual Studio Code Extension
3. Docker Extension

For information on contributing or submitting issues see the Visual Studio GitHub Repository. Visual Studio Code documentation is also Open Source, and you can contribute or submit issues from the Visual Studio Documentation GitHub Repository.

Raspberry Pi Hardware

If you are attending a workshop, then you can use a shared network-connected Raspberry Pi. You can also use your own network-connected Raspberry Pi for this hands-on lab.

Shared Raspberry Pi

If you are attending a workshop and using a shared Raspberry Pi, then you will need the following information from the lab instructor.

1. The **Network IP Address** of the Raspberry Pi
2. Your assigned **login name** and **password**.

Personal Raspberry Pi

If you are using your own network-connected Raspberry Pi, then you need:

1. The Raspberry Pi **Network IP Address**, the **login name**, and **password**.
2. You need to run the following commands on your Raspberry Pi to set two environment variables required for the hands-on lab.

```
echo "export LAB_PORT=\$(shuf -i 5000-8000 -n 1)" >> ~/.bashrc
echo "export LAB_HOST=\$(hostname -I | cut -d' ' -f 1)" >>  ~/.bashrc
source .bashrc
```

SSH Authentication with private/public keys

Setting up a public/private key pair for SSH authentication is a secure and fast way to authenticate from your computer to the Raspberry Pi. This is needed for this hands-on lab.

SSH for Linux and macOS

From a Linux or macOS **Terminal Console** run the following commands:

1. Create your key. This is typically a one-time operation. **Take the default options**.

   ```
   ssh-keygen -t rsa -b 4096 -f ~/.ssh/id_rsa_python_lab
   ```

2. Copy the public key to the Raspberry Pi.

   ```
   ssh-copy-id -i ~/.ssh/id_rsa_python_lab <login@Raspberry IP Address>
   ```

   For example:

```
ssh-copy-id -i ~/.ssh/id_rsa_python_lab dev99@192.168.1.200
```

3. Test the SSH Authentication Key

   ```
   ssh -i ~/.ssh/id_rsa_python_lab <login@Raspberry IP Address>
   ```

   A new SSH session will start. You should now be connected to the Raspberry Pi **without** being prompted for the password.
4. Close the SSH session. In the SSH terminal, type exit, followed by ENTER.

SSH for Windows 10 (1809+) Users with PowerShell

1. Start PowerShell as Administrator and install OpenSSH.Client

   ```
       Add-WindowsCapability -Online -Name OpenSSH.Client
   ```

2. **Exit** PowerShell
3. Restart PowerShell (**NOT** as Administrator)
4. Create an SSH Key

   ```
   ssh-keygen -t rsa -f $env:userprofile\.ssh\id_rsa_python_lab
   ```

5. Copy SSH Key to Raspberry Pi

   ```
   cat $env:userprofile\.ssh\id_rsa_python_lab.pub | ssh `
   <login@Raspberry IP Address> `
   "mkdir -p ~/.ssh; cat >> ~/.ssh/authorized_keys"
   ```

6. Test the SSH Authentication Key

   ```
   ssh -i $env:userprofile\.ssh\id_rsa_python_lab <login@Raspberry IP Address>
   ```

   A new SSH session will start. You should now be connected to the Raspberry Pi **without** being prompted for the password.
7. Close the SSH session. In the SSH terminal, type exit, followed by ENTER.

SSH for earlier versions of Windows

SSH for earlier versions of Windows

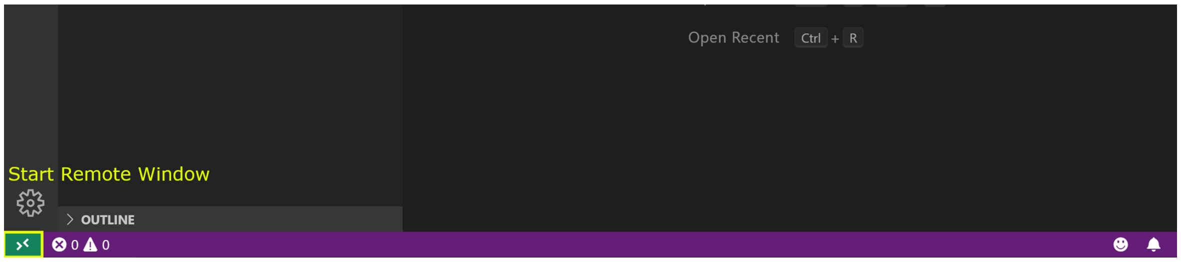
Trouble Shooting SSH Client Installation

- Remote Development using SSH
- Installing a supported SSH client

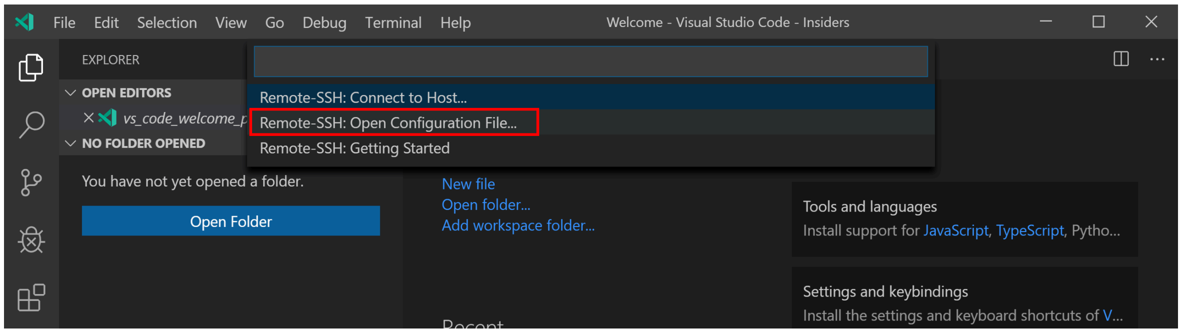
Configure Visual Studio Code Remote SSH Development

Configure Visual Studio Code **Remote SSH** with the Raspberry Pi **Network IP Address**, **login name**, and **SSH key file** you will be using for the hands-on lab.

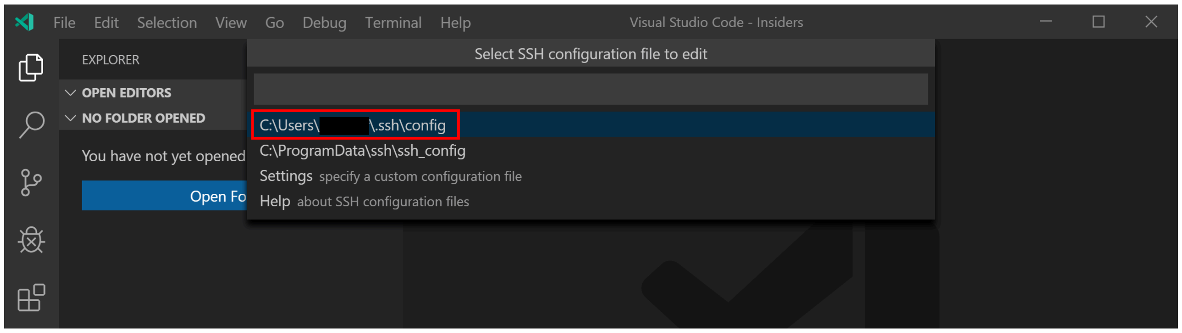
1. Start Visual Studio Code Insiders Edition
2. Click the **Open Remote Windows** button. You will find this button in the bottom left-hand corner of the Visual Studio Code window.



3. Select **Open Configuration File**

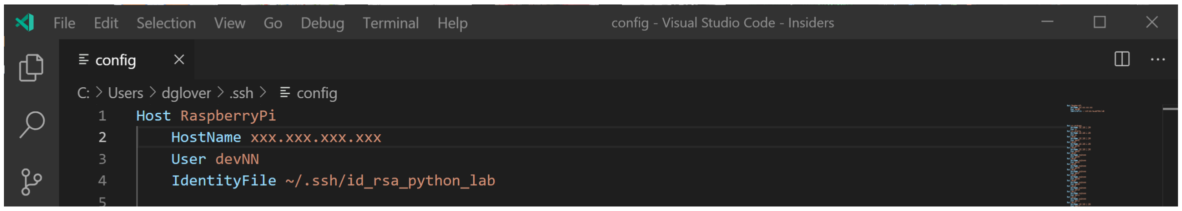

File Edit Selection View Go Debug Terminal Help
Welcome - Visual Studio Code - Insiders
EXPLORER
OPEN EDITORS
vs_code_welcome_p
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
Remote-SSH: Connect to Host...
Remote-SSH: Open Configuration File...
Remote-SSH: Getting Started
New file
Open folder...
Add workspace folder...
Recent
Tools and languages
Install support for JavaScript, TypeScript, Pytho...
Settings and keybindings
Install the settings and keyboard shortcuts of V...

4. Select the user .ssh config file


File Edit Selection View Go Debug Terminal Help
Visual Studio Code - Insiders
EXPLORER
OPEN EDITORS
NO FOLDER OPENED
You have not yet opened
Open Fo
Select SSH configuration file to edit
C:\Users\[redacted]\.ssh\config
C:\ProgramData\ssh\ssh_config
Settings specify a custom configuration file
Help about SSH configuration files

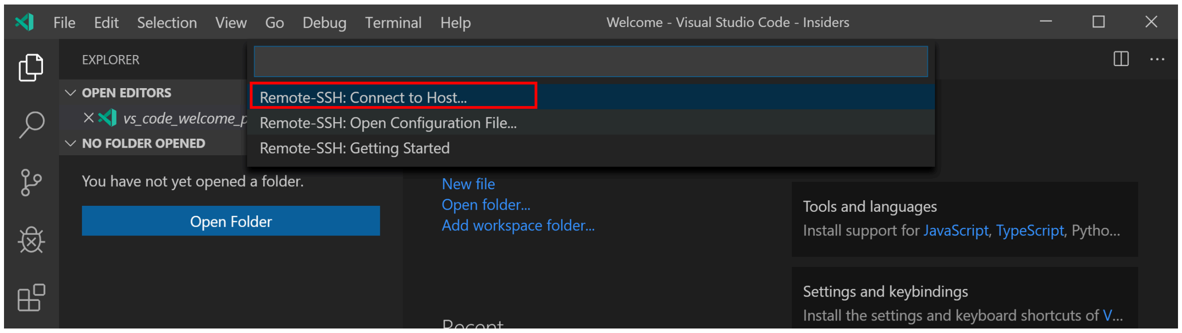
5. Set the SSH connection configuration. You need the Raspberry Pi **IP Address**, the Raspberry Pi **login name**, and finally set the **IdentityFile** field to **~/.ssh/**

id_rsa_python_lab. Save these changes (Ctrl+S).

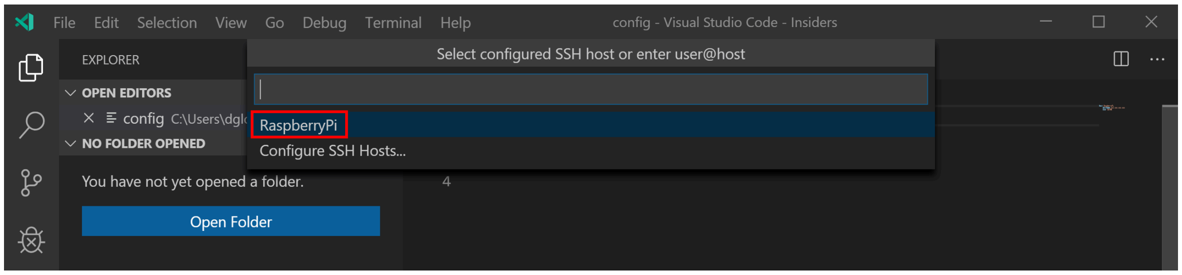

File Edit Selection View Go Debug Terminal Help
config - Visual Studio Code - Insiders
config
C: > Users > dglover > .ssh > config

```
Host RaspberryPi
    HostName xxx.xxx.xxx.xxx
    User devNN
    IdentityFile ~/.ssh/id_rsa_python_lab
```

6. Click the Open Remote Windows button (bottom left) then select **Remote SSH: Connect to Host**

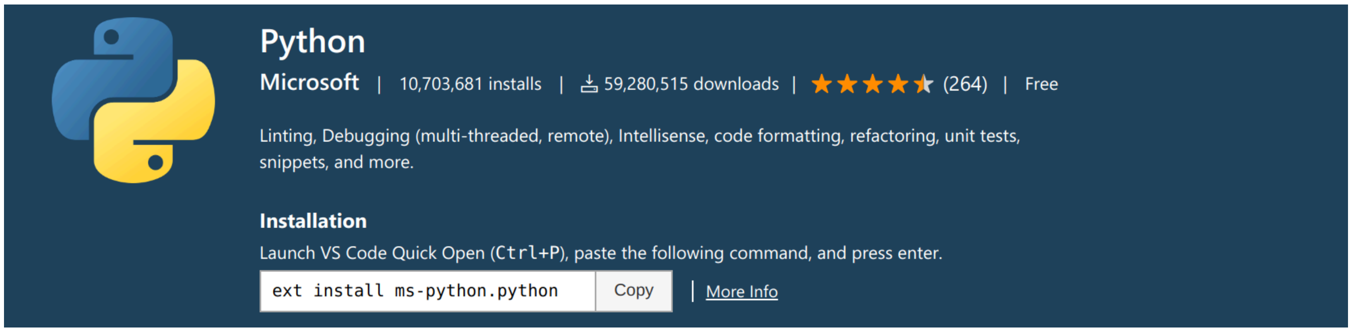

File Edit Selection View Go Debug Terminal Help
Welcome - Visual Studio Code - Insiders
EXPLORER
OPEN EDITORS
vs_code_welcome_p
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
Remote-SSH: Connect to Host...
Remote-SSH: Open Configuration File...
Remote-SSH: Getting Started
New file
Open folder...
Add workspace folder...
Recent
Tools and languages
Install support for JavaScript, TypeScript, Pytho...
Settings and keybindings
Install the settings and keyboard shortcuts of V...

7. Select the host **RaspberryPi** configuration


File Edit Selection View Go Debug Terminal Help
config - Visual Studio Code - Insiders
EXPLORER
OPEN EDITORS
config C:\Users\dgl
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
Select configured SSH host or enter user@host
RaspberryPi
Configure SSH Hosts...

It will take a moment to connect to the Raspberry Pi.

Install the Python Visual Studio Code Extension


Python
Microsoft | 10,703,681 installs | 59,280,515 downloads | ★★★★☆ (264) | Free

Linting, Debugging (multi-threaded, remote), Intellisense, code formatting, refactoring, unit tests, snippets, and more.

Installation

Launch VS Code Quick Open (Ctrl+P), paste the following command, and press enter.

```
ext install ms-python.python
```

Launch Visual Studio Code Quick Open (Ctrl+P), paste the following command, and press enter:

```
ext install ms-python.python
```

See the Python Extension page for information about using the extension.

Open the Lab2 Docker Debug Project

From **Visual Studio Code**, select **File** from the main menu, then **Open Folder**. Navigate to and open the **github/lab2-docker-debug** folder.

1. From VS Code: File -> Open Folder, navigate to **github/lab2-docker-debug**.
2. Expand the App folder and open the app.py file.

Creating an Azure IoT Central Application

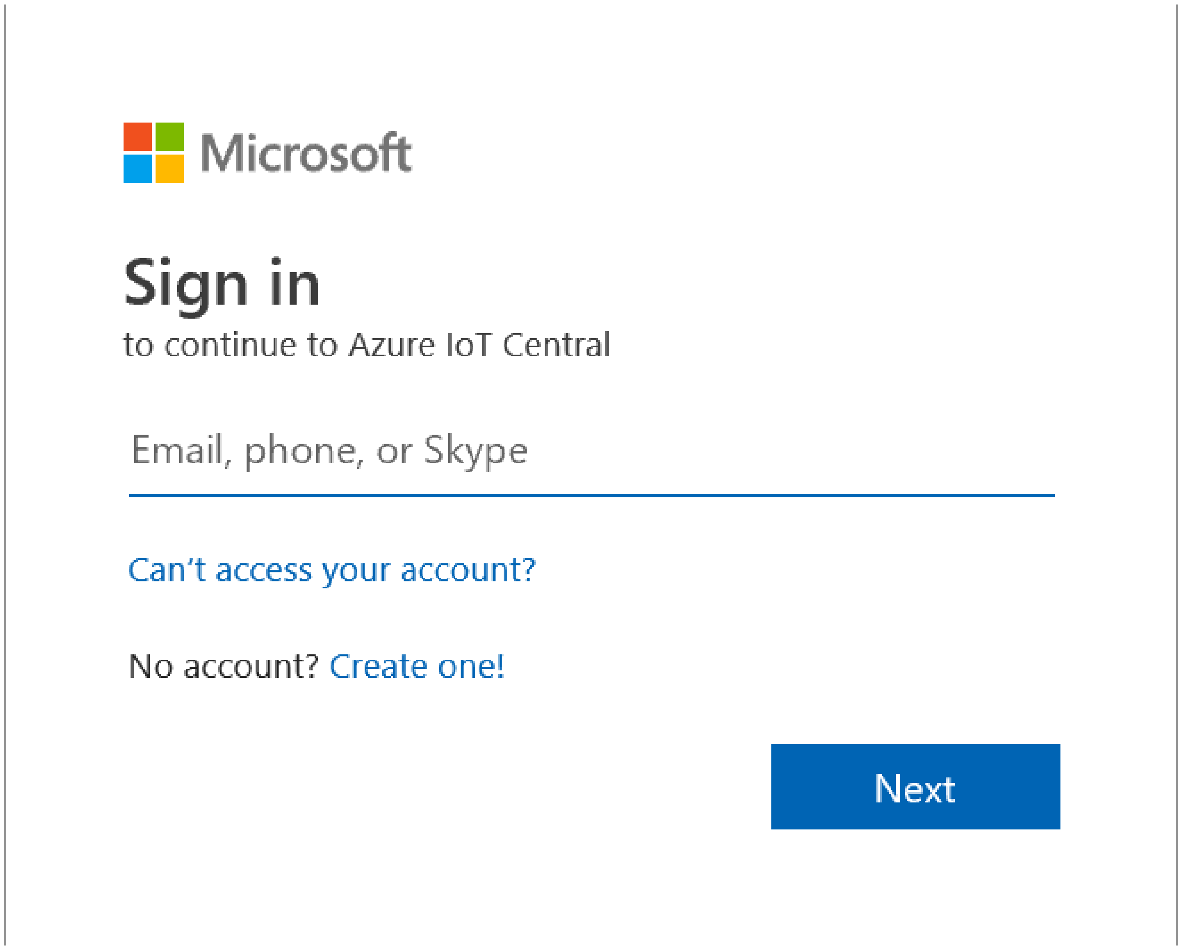
We are going to create an Azure IoT Central application, then a device, and finally a device **connection string** needed for the application that will run in the Docker container.

As a *builder*, you use the Azure IoT Central UI to define your Microsoft Azure IoT Central application. This quickstart shows you how to create an Azure IoT Central application that contains a sample *device template* and simulated *devices*.

Create a New IoT Central Application

1. Open the Azure IoT Central in a new browser tab, then click **Getting started**.
2. Next, you'll need to sign with your **Microsoft** Personal, or Work, or School account. If

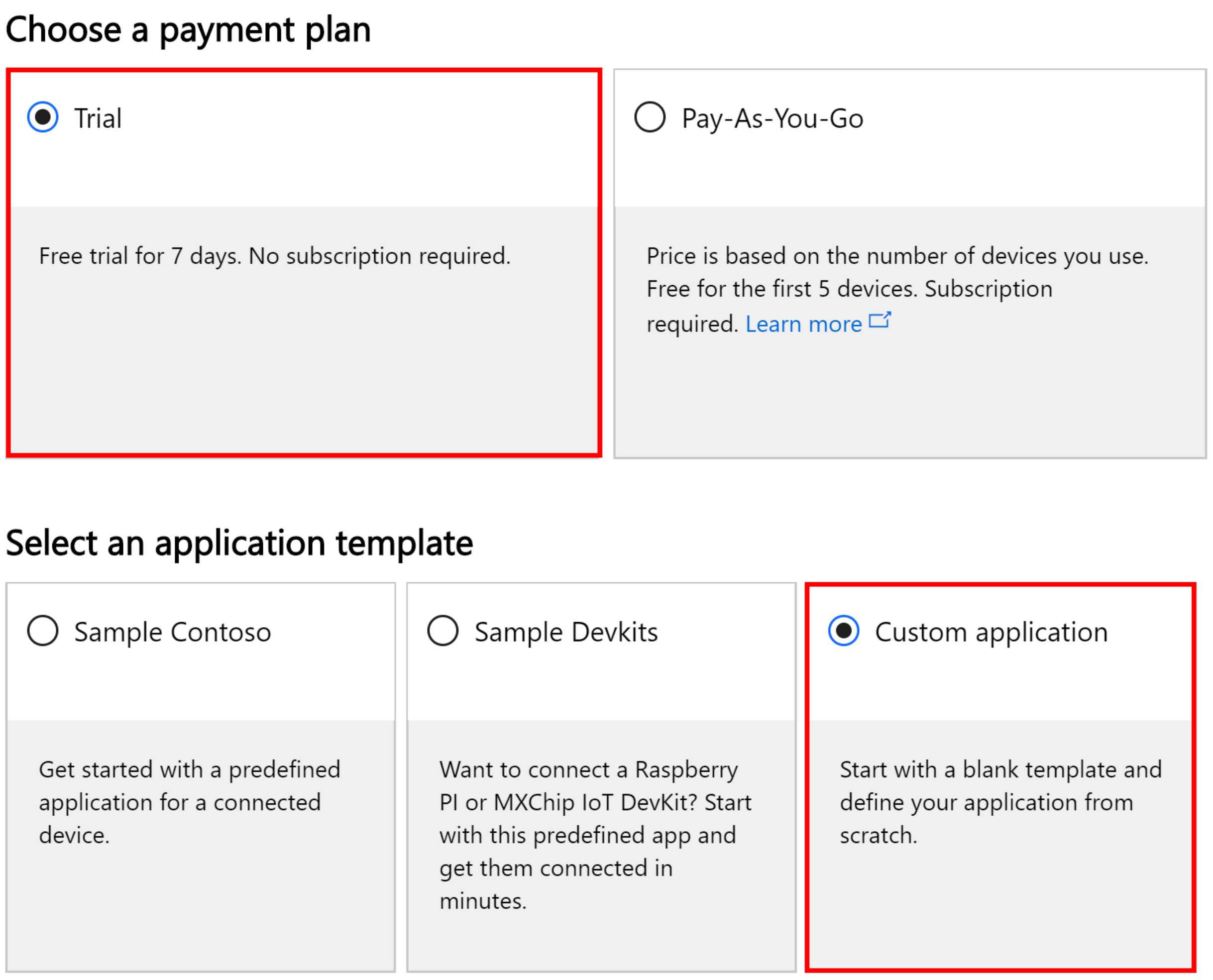
you do not have a Microsoft account, then you can create one for free using the **Create one!** link.


Microsoft

Sign in

to continue to Azure IoT Central

3. Create a new Azure IoT Central application, select **New Application**. This takes you to the **Create Application** page.
4. Select **Trail**, **Custom application**, name your IoT Central application and complete the sign-up information.



Choose a payment plan

[✓ Trial]

Free trial for 7 days. No subscription required.

Pay-As-You-Go

Price is based on the number of devices you use. Free for the first 5 devices. Subscription required. Learn more

Select an application template

Sample Contoso

Get started with a predefined application for a connected device.

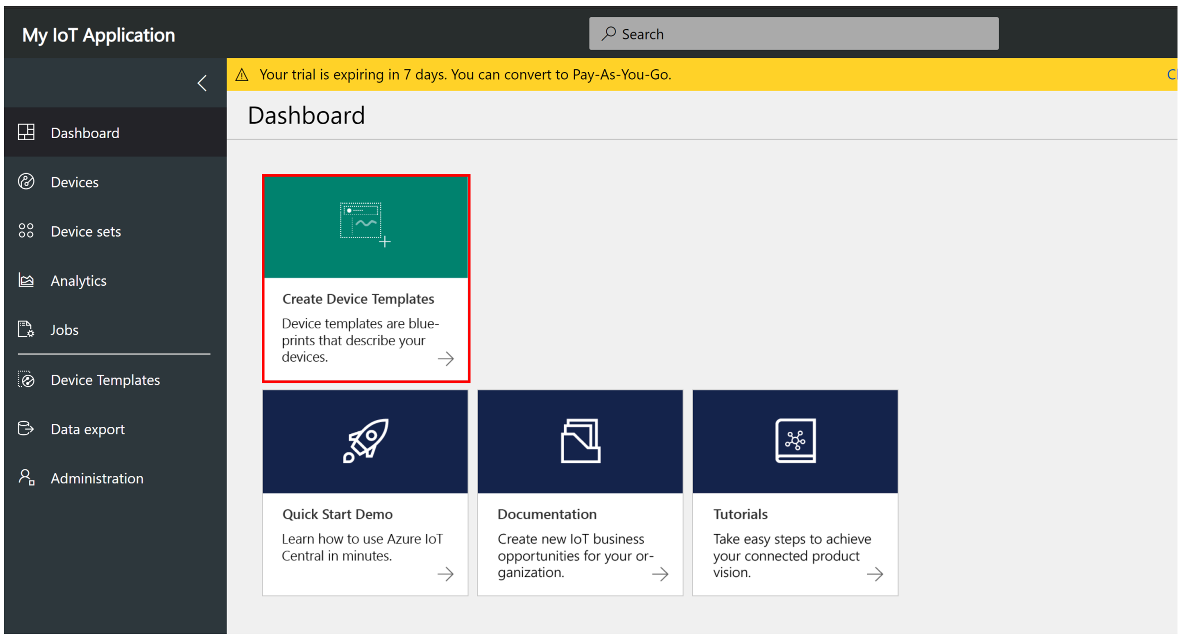
Sample Devkits

Want to connect a Raspberry PI or MXChip IoT DevKit? Start with this predefined app and get them connected in minutes.

[✓ Custom application]

Start with a blank template and define your application from scratch.

4. Click **Create Device Templates**, then select the **Custom** template, name your template, for example, **Raspberry**. Then click Create



My IoT Application

Search

Your trial is expiring in 7 days. You can convert to Pay-As-You-Go.

Dashboard

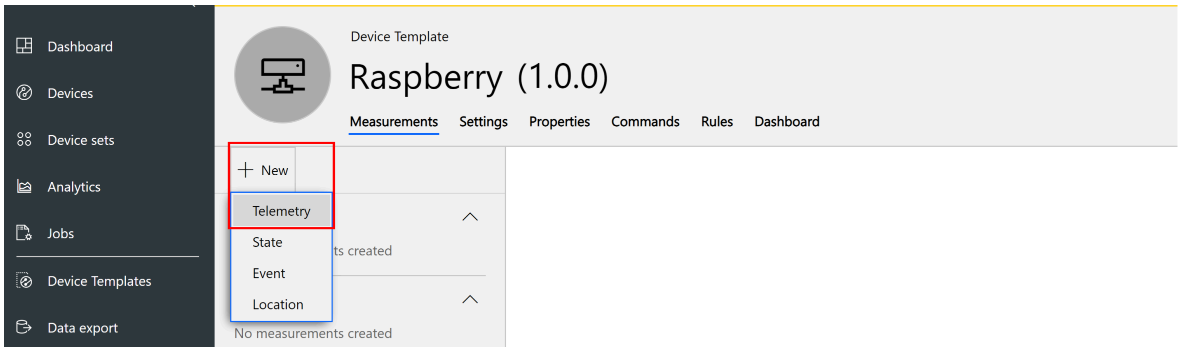
Create Device Templates
Device templates are blue-prints that describe your devices.

Quick Start Demo
Learn how to use Azure IoT Central in minutes.

Documentation
Create new IoT business opportunities for your organization.

Tutorials
Take easy steps to achieve your connected product vision.

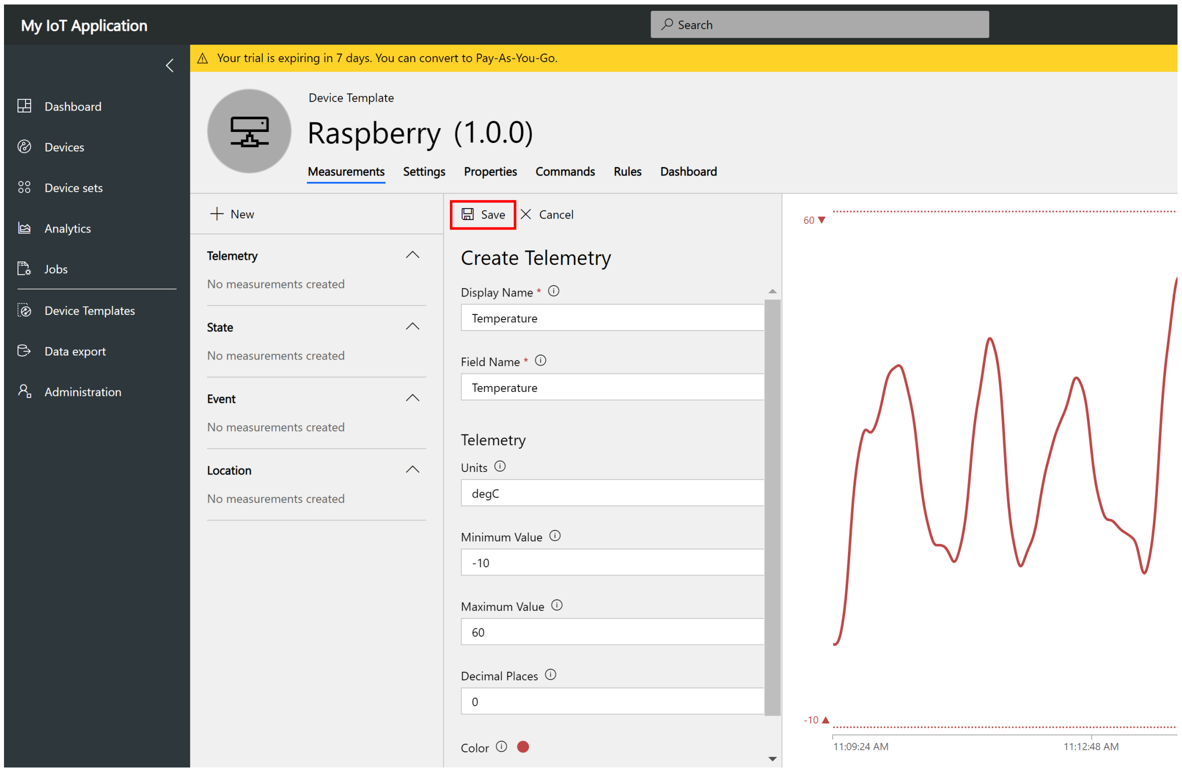
5. Edit the Template, add **Measurements** for **Temperature**, **Humidity**, and **Pressure** telemetry.
   Measurements are the data that comes from your device. You can add multiple measurements to your device template to match the capabilities of your device.
   - **Telemetry** measurements are the numerical data points that your device collects over time. They're represented as a continuous stream. An example is temperature.
   - **Event** measurements are point-in-time data that represents something of significance on the device. A severity level represents the importance of an event. An example is a fan motor error.
   - **State** measurements represent the state of the device or its components over a period of time. For example, a fan mode can be defined as having Operating and Stopped as the two possible states.
   - **Location** measurements are the longitude and latitude coordinates of the device over a period of time in. For example, a fan can be moved from one location to another.



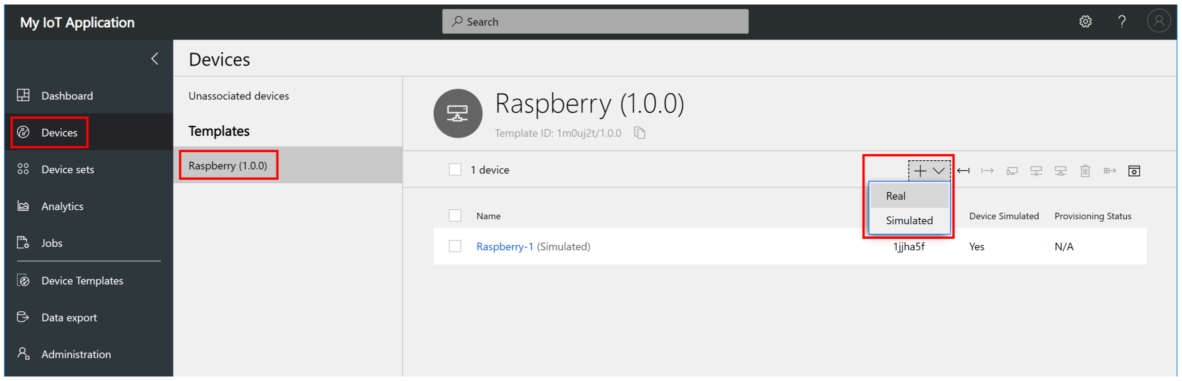
Use the information in the following table to set up three telemetry measurements. The field name is case-sensitive.
You **must** click **Save** after each measurement is defined.

Display Name	Field name	Units	Minimum	Maximum	Decimals
Humidity	Humidity	%	0	100	0
Temperature	Temperature	degC	-10	60	0
Pressure	Pressure	hPa	800	1260	0

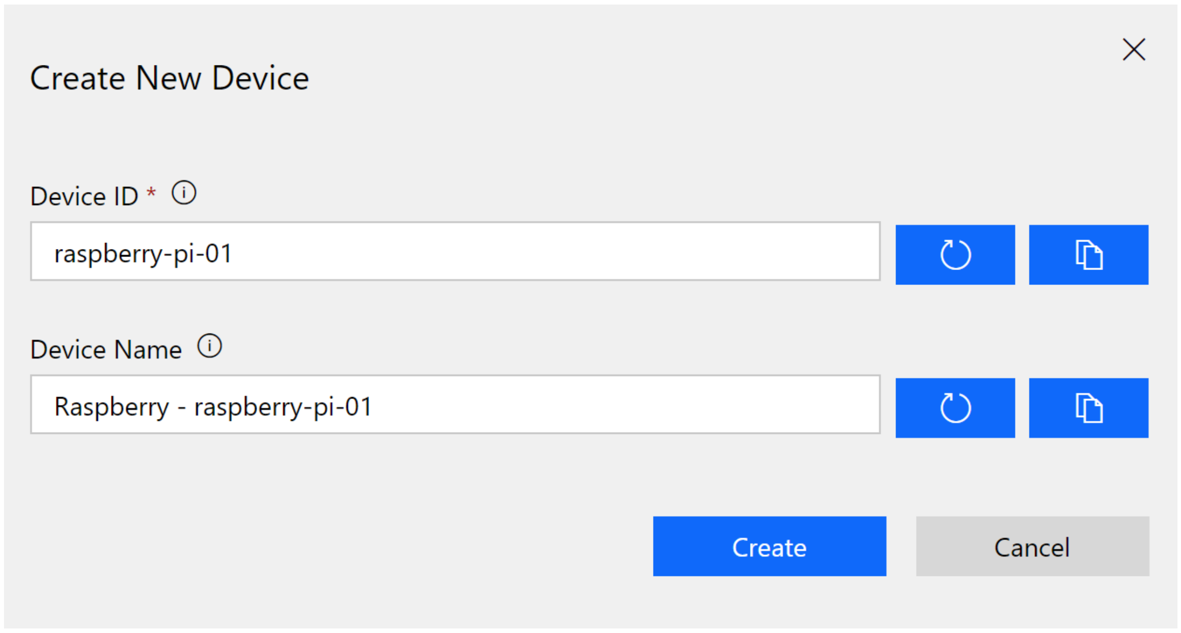
The following is an example of setting up the **Temperature** telemetry measurement.



6. Click **Device** on the sidebar menu, select the **Raspberry** template you created. IoT central supports real devices, such as the Raspberry Pi used for this lab, as well as simulated devices which generate random data useful for system testing.
7. Select **Real**.



Name your **Device ID** so you can easily identify the device in the IoT Central portal, then click **Create**.

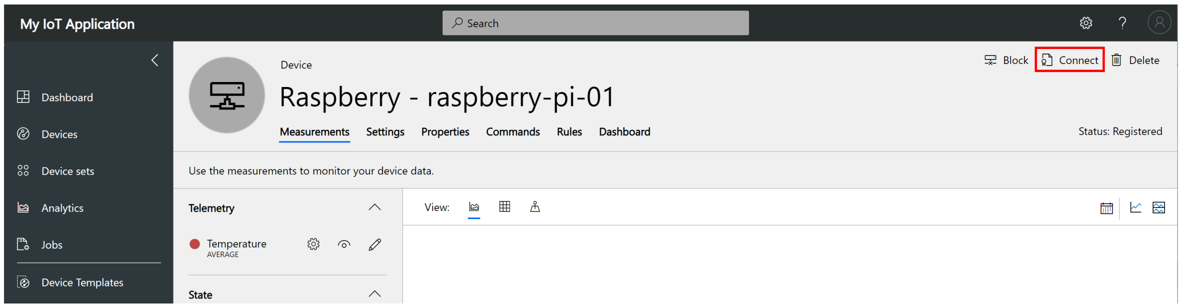


Create New Device

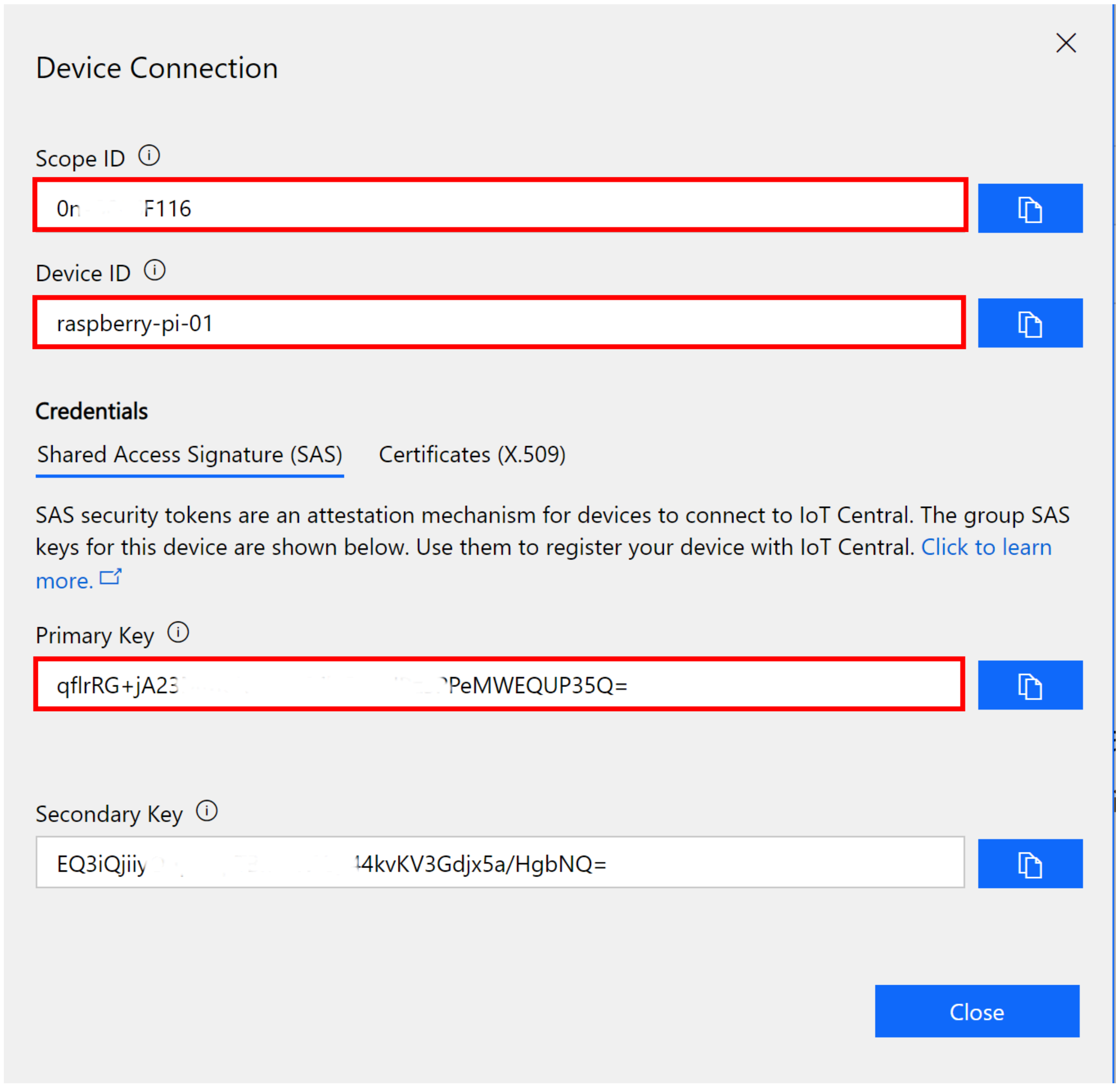
Device ID * : raspberry-pi-01

Device Name: Raspberry - raspberry-pi-01

8. When you have created your real device click the **Connect** button in the top right-hand corner of the screen to display the device credentials.



Leave this page open as you will need this connection information for the next step in the hands-on lab.


Device Connection

Scope ID

0n F116

Device ID

raspberry-pi-01

Credentials

Shared Access Signature (SAS) Certificates (X.509)

SAS security tokens are an attestation mechanism for devices to connect to IoT Central. The group SAS keys for this device are shown below. Use them to register your device with IoT Central. Click to learn more.

Primary Key

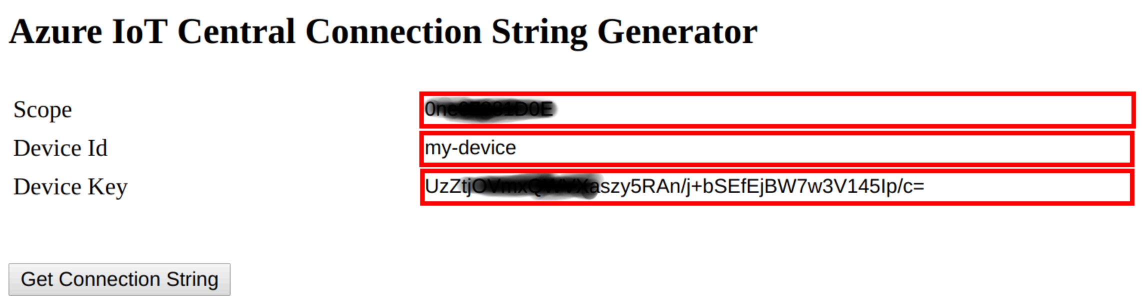
qflrRG+jA23 PeMWEQUP35Q=

Secondary Key

EQ3iQjiiy 4kvKV3Gdjx5a/HgbNQ=

Generate an Azure IoT Hub Connection String

1. Hold the control key down and click the following link Connection String Generator to open in a new tab.
   Copy and paste the **Scope Id**, **Device Id**, and the **Primary Key** from the Azure IoT Central Device Connection panel to the Connection String Generator page and click **Get Connection String**.


Azure IoT Central Connection String Generator

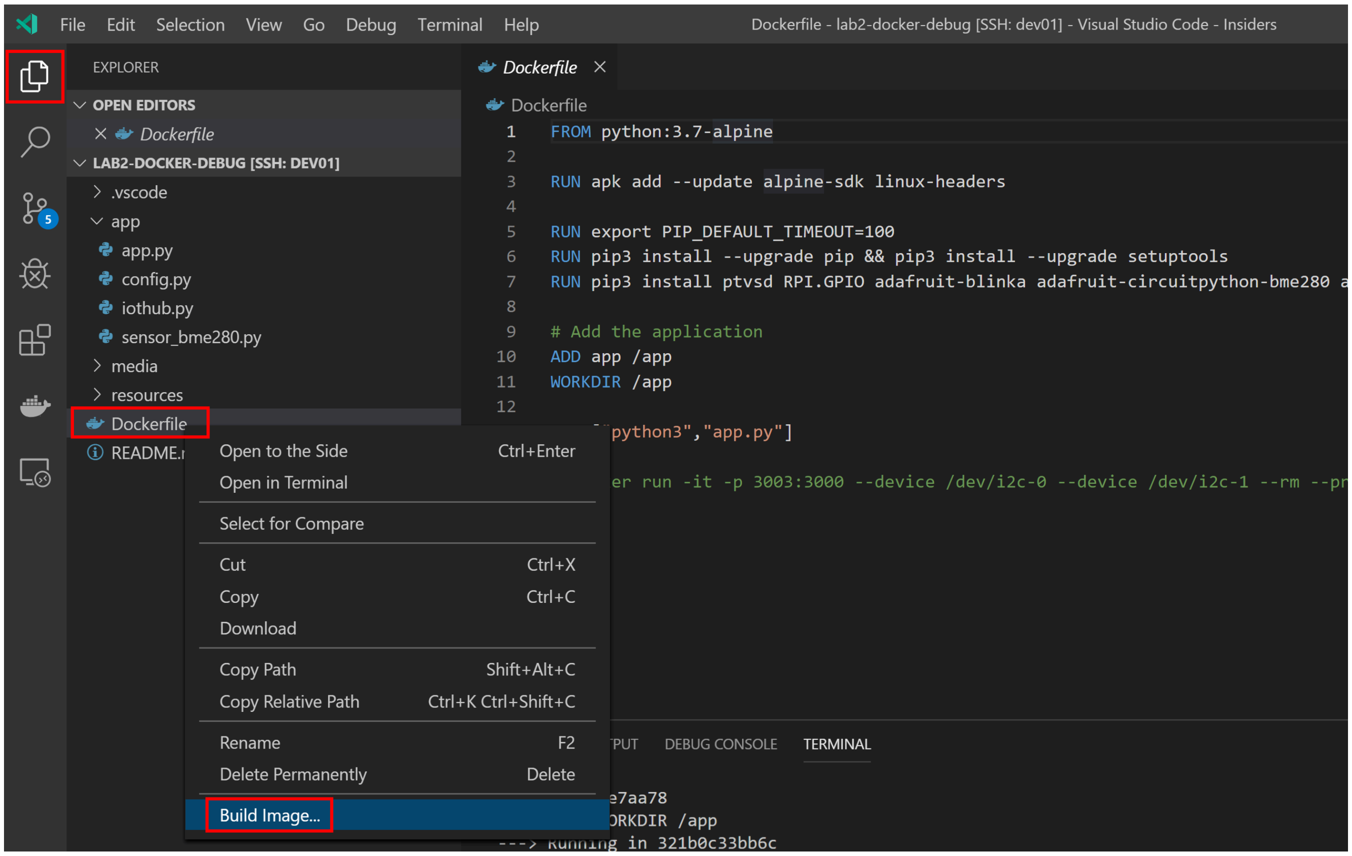
Scope	0ne[redacted]D0E
Device Id	my-device
Device Key	UzZtjO[redacted]aszy5RAn/j+bSEfEjBW7w3V145Ip/c=

2. Copy the generated connection string to the clipboard as you will need it for the next step.

Open the Visual Studio Code Docker Debugging Lab

Build the Docker Image

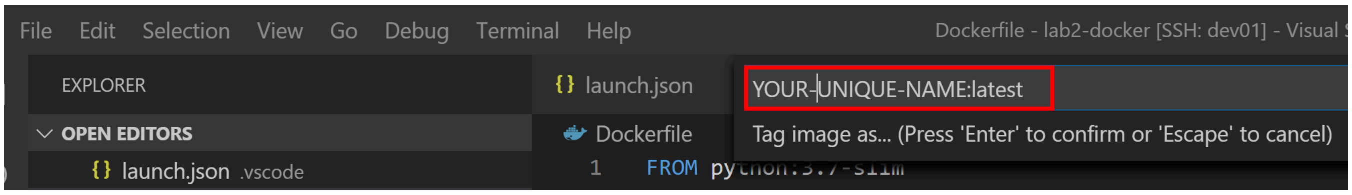
1. Switch back to the project you opened with Visual Studio Code. Open the **env-file** (environment file). This file will contain environment variables that will be passed into the Docker container.
2. Paste the connection string you copied in the previous step into the env-file on the same line, and after **CONNECTION_STRING=**.
   For example:
   ```
   CONNECTION_STRING=HostName=saas-iothub-8135cd3b....
   ```
3. Save the env-file file (Ctrl+S)
4. Ensure **Explorer** selected in the activity bar, right mouse click file named **Dockerfile** and select **Build Image**.


File Edit Selection View Go Debug Terminal Help
Dockerfile - lab2-docker-debug [SSH: dev01] - Visual Studio Code - Insiders
EXPLORER
OPEN EDITORS
Dockerfile
LAB2-DOCKER-DEBUG [SSH: DEV01]
.vscode
app
app.py
config.py
iothub.py
sensor_bme280.py
media
resources
Dockerfile
README.
Open to the Side Ctrl+Enter
Open in Terminal
Select for Compare
Cut Ctrl+X
Copy Ctrl+C
Download
Copy Path Shift+Alt+C
Copy Relative Path Ctrl+K Ctrl+Shift+C
Rename F2
Delete Permanently Delete
Build Image...
PUT DEBUG CONSOLE TERMINAL

```
FROM python:3.7-alpine

RUN apk add --update alpine-sdk linux-headers

RUN export PIP_DEFAULT_TIMEOUT=100
RUN pip3 install --upgrade pip && pip3 install --upgrade setuptools
RUN pip3 install ptvsd RPI.GPIO adafruit-blinka adafruit-circuitpython-bme280 a

# Add the application
ADD app /app
WORKDIR /app

    "python3","app.py"]

    er run -it -p 3003:3000 --device /dev/i2c-0 --device /dev/i2c-1 --rm --pr
```

```
e7aa78
ORKDIR /app
---> Running in 321b0c33bb6c
```

5. Give your docker build image a **unique name** - eg the first part of your email address, your nickname, something memorable, followed by **:latest**. The name needs to be unique otherwise it will clash with others building Docker images on the same Raspberry Pi.
   For example **glovebox:latest**

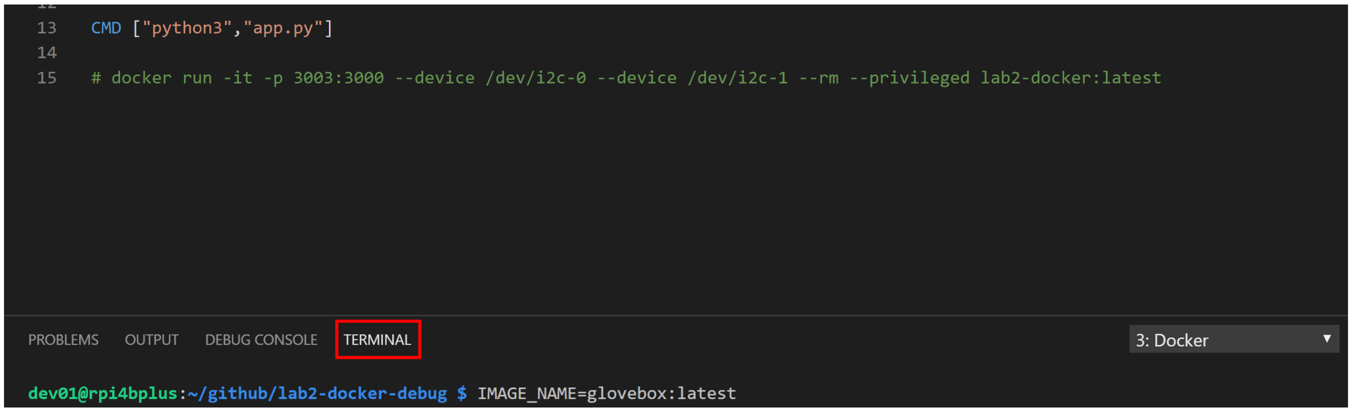

File Edit Selection View Go Debug Terminal Help
Dockerfile - lab2-docker [SSH: dev01] - Visual
EXPLORER
OPEN EDITORS
launch.json .vscode
launch.json
Dockerfile

```
YOUR-UNIQUE-NAME:latest
Tag image as... (Press 'Enter' to confirm or 'Escape' to cancel)
```

```
FROM python:3.7-slim
```

Run the Docker Image

Paste these commands into the Visual Studio Code Terminal Window. You can open a new Terminal Window with **Ctrl+Shift+`**



```
CMD ["python3","app.py"]

# docker run -it -p 3003:3000 --device /dev/i2c-0 --device /dev/i2c-1 --rm --privileged lab2-docker:latest
```

```
dev01@rpi4bplus:~/github/lab2-docker-debug $ IMAGE_NAME=glovebox:latest
```

1. Set the IMAGE_NAME environment variable. This is the unique name you used when you built the Docker Image.

```
export IMAGE_NAME=<YOUR-UNIQUE-NAME>:latest
```

2. Display and make a note of your $LAB_PORT environment variable. This is set in the .bashrc file. You will need this for the debugger step.

```
echo -e "\e[7mYour Lab Port is $LAB_PORT\e[0m"
```

3. Start the Docker Container

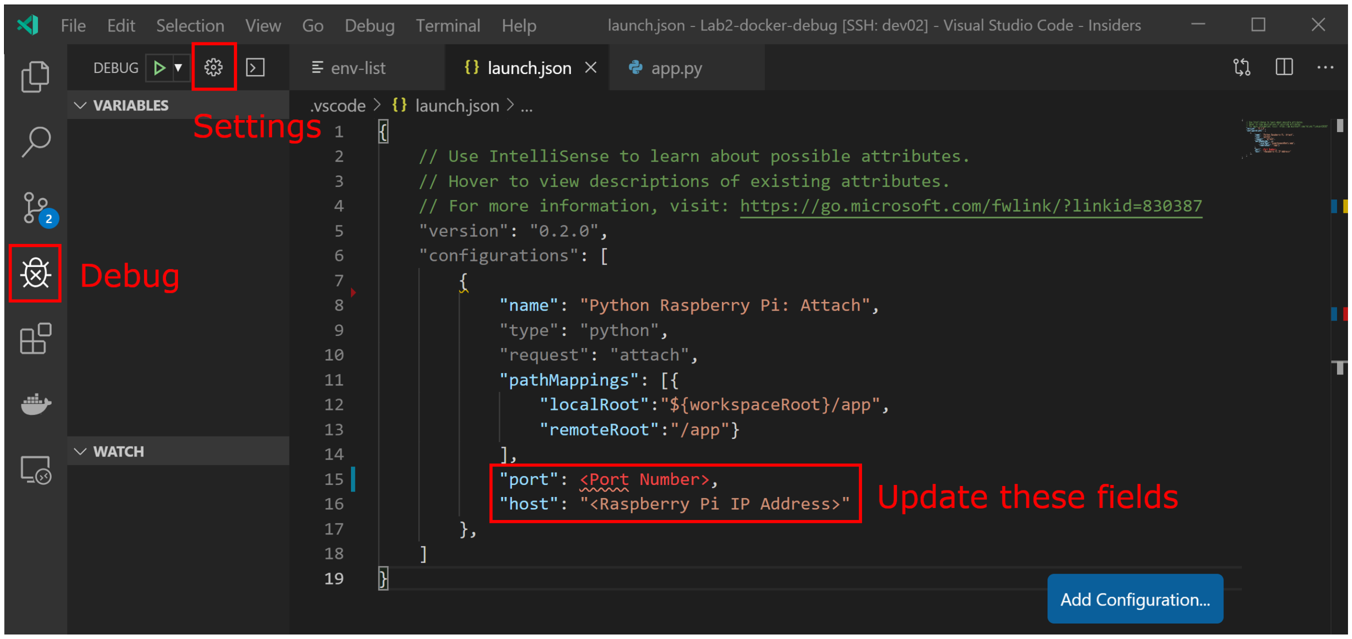
```
docker run -it \
-p $LAB_PORT:3000 \
-v /tmp/sensor.lock:/tmp/sensor.lock \
--env-file ~/github/Lab2-docker-debug/env-list \
--device /dev/i2c-0 --device /dev/i2c-1 \
--rm --privileged $IMAGE_NAME
```

The **Docker run** will start your container in interactive mode (**--it**), (**-p**) will map the **$LAB_PORT** to port 3000 in the container, the BME280 sensor is connected to the host I2C bus, (**--device**) maps the host I2C bus into the container, (**--rm**) removes the container when you stop it, (**--privileged**) grants elevated permissions to the container so that is can access the host I2C bus from within the container, and finally Docker starts the image you built/named.

Configure the Visual Studio Code Debugger

1. Click **Debug** on the Visual Studio activity bar.

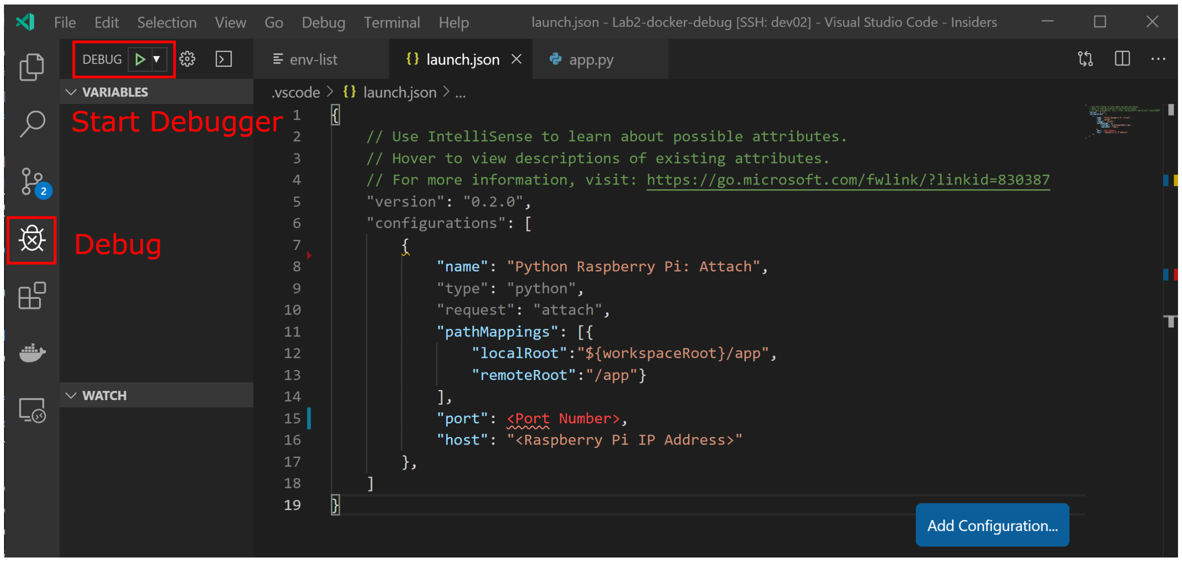
2. Clicking **Settings**
3. Update the **port** value to the **Port Number** displayed when you started the Docker Container in the Visual Studio Terminal Window.
4. Update the **host** to match your Raspberry Pi IP Address.


File Edit Selection View Go Debug Terminal Help
launch.json - Lab2-docker-debug [SSH: dev02] - Visual Studio Code - Insiders
DEBUG
env-list
launch.json
app.py
VARIABLES
WATCH
.vscode > launch.json > ...
Settings
Debug
Update these fields
Add Configuration...

```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Python Raspberry Pi: Attach",
            "type": "python",
            "request": "attach",
            "pathMappings": [{
                "localRoot":"${workspaceRoot}/app",
                "remoteRoot":"/app"}
            ],
            "port": <Port Number>,
            "host": "<Raspberry Pi IP Address>"
        },
    ]
}
```

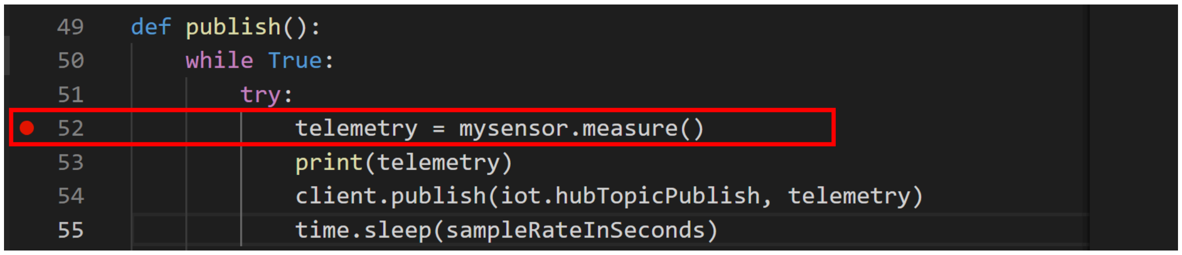
Attach the Debugger to the Docker Container

1. Click **Debug** on the Visual Studio activity bar.
2. Click the **Debug Selection** dropdown, and ensure **Python Raspberry Pi: Attach** is selected.
3. Click the **Run** button to attach the debugger.


File Edit Selection View Go Debug Terminal Help
launch.json - Lab2-docker-debug [SSH: dev02] - Visual Studio Code - Insiders
DEBUG
env-list
launch.json
app.py
VARIABLES
WATCH
.vscode > launch.json > ...
Start Debugger
Debug
Add Configuration...

```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Python Raspberry Pi: Attach",
            "type": "python",
            "request": "attach",
            "pathMappings": [{
                "localRoot":"${workspaceRoot}/app",
                "remoteRoot":"/app"}
            ],
            "port": <Port Number>,
            "host": "<Raspberry Pi IP Address>"
        },
    ]
}
```

4. From **Explorer** on the Visual Studio Code activity bar, open the **app.py** file
5. Set a breakpoint in the while True loop.



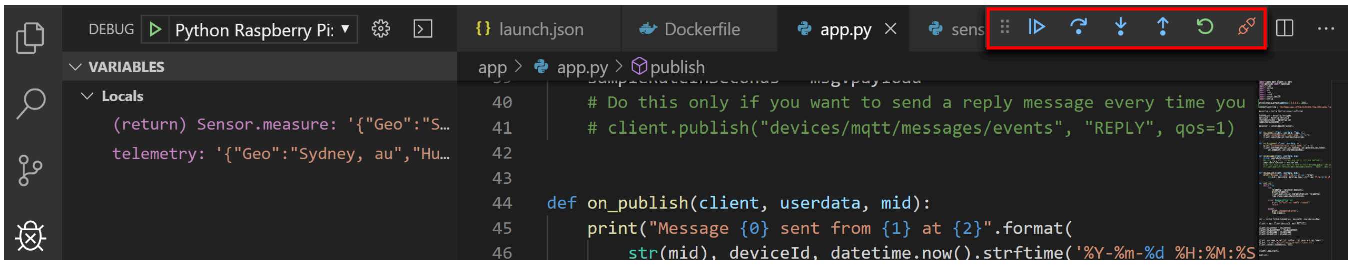
```
def publish():
    while True:
        try:
            telemetry = mysensor.measure()
            print(telemetry)
            client.publish(iot.hubTopicPublish, telemetry)
            time.sleep(sampleRateInSeconds)
```

Ensure the **app.py** file is open, set a breakpoint at line **64**, in the **publish** function (**telemetry = mysensor.measure()**) by doing any one of the following:

   - With the cursor on that line, press F9, or,
   - With the cursor on that line, select the Debug > Toggle Breakpoint menu command, or, click directly in the margin to the left of the line number (a faded red dot appears when hovering there). The breakpoint appears as a red dot in the left margin:

Debugger Controls

Debugger Controls allow for Starting, Pausing, Stepping in to, Stepping out off, restarting code, and finally Disconnecting the debugger.



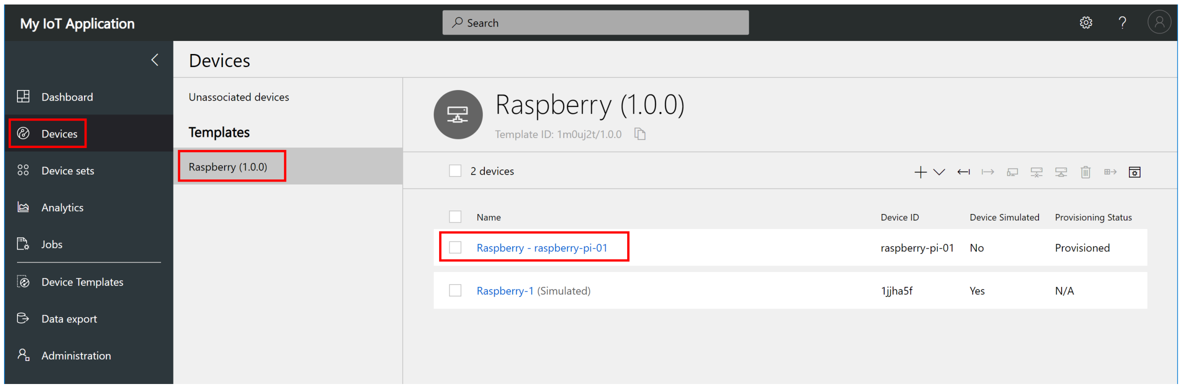
```
VARIABLES
  Locals
    (return) Sensor.measure: '{"Geo":"S…
    telemetry: '{"Geo":"Sydney, au","Hu…
```

```
        # Do this only if you want to send a reply message every time you
        # client.publish("devices/mqtt/messages/events", "REPLY", qos=1)


def on_publish(client, userdata, mid):
    print("Message {0} sent from {1} at {2}".format(
        str(mid), deviceId, datetime.now().strftime('%Y-%m-%d %H:%M:%S
```

1. Explore the **Debug Console** (Ctrl+Shift+Y)
   1. With debugger stopped at the breakpoint in the **publish** function, explore the **Variables Window** and try typing **print(telemetry** into the debug prompt.
2. Explore the **Terminal**
3. From the Debugger Toolbar, **Disconnect** the debugger so the application in the Docker container continues to run and stream telemetry to **Azure IoT Central**.

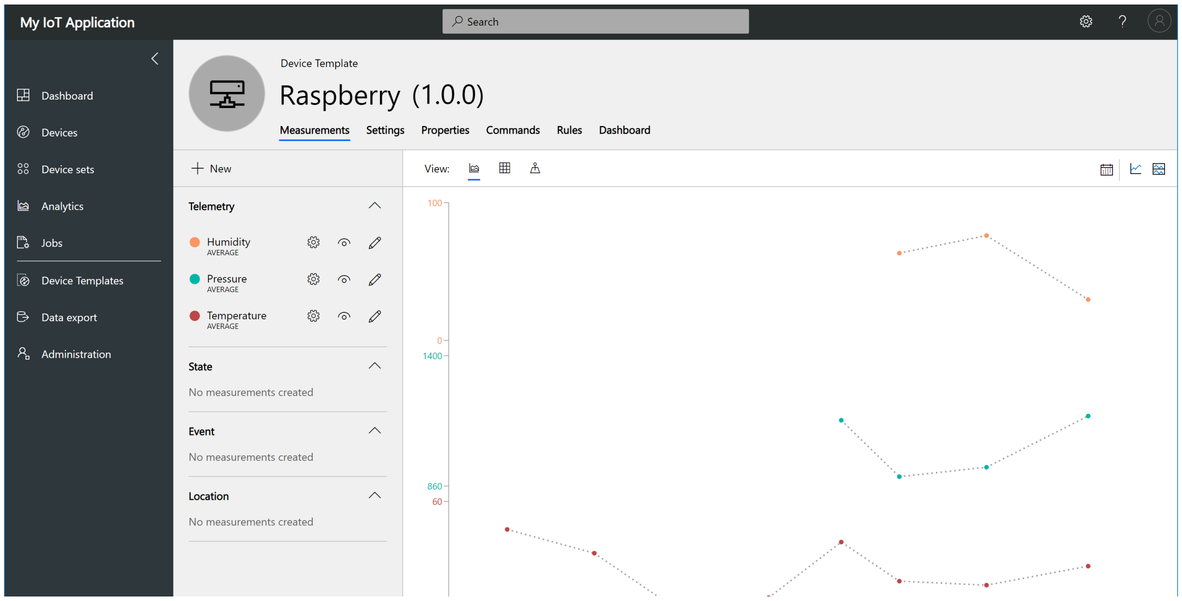
Exploring Device Telemetry in Azure IoT Central

1. Use **Device** to navigate to the **Measurements** page for the real Raspberry Pi device you added:



Name	Device ID	Device Simulated	Provisioning Status
Raspberry - raspberry-pi-01	raspberry-pi-01	No	Provisioned
Raspberry-1 (Simulated)	1jjha5f	Yes	N/A

2. On the **Measurements** page, you can see the telemetry streaming from the Raspberry Pi device:



Finished

Complete. Congratulations

Appendix

Azure IoT Central

Take a tour of the Azure IoT Central UI

This article introduces you to the Microsoft Azure IoT Central UI. You can use the UI to create, manage, and use an Azure IoT Central solution and its connected devices.

As a *builder*, you use the Azure IoT Central UI to define your Azure IoT Central solution. You can use the UI to:

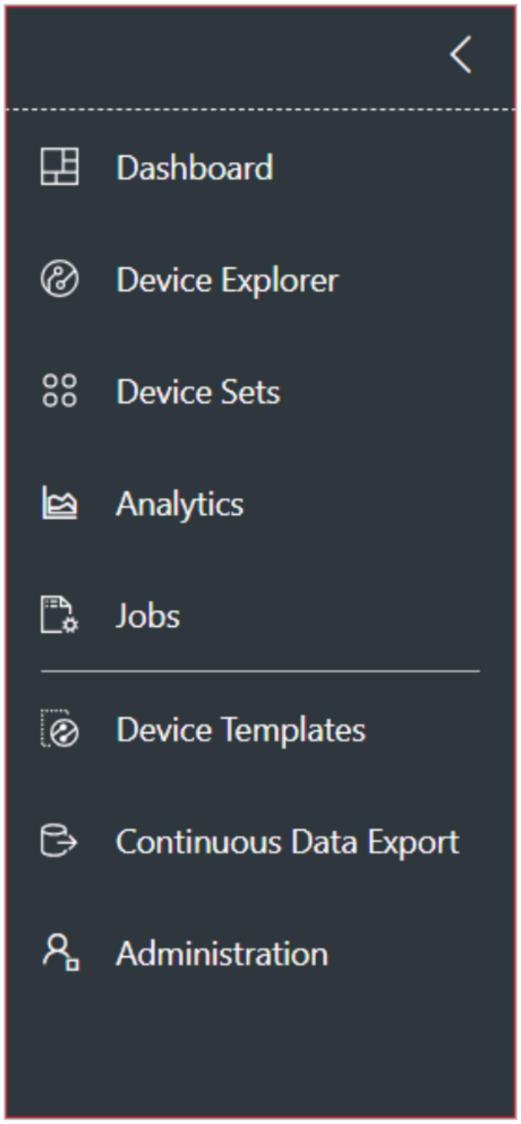
- Define the types of device that connect to your solution.
- Configure the rules and actions for your devices.
- Customize the UI for an *operator* who uses your solution.

As an *operator*, you use the Azure IoT Central UI to manage your Azure IoT Central solution. You can use the UI to:

- Monitor your devices.
- Configure your devices.
- Troubleshoot and remediate issues with your devices.
- Provision new devices.

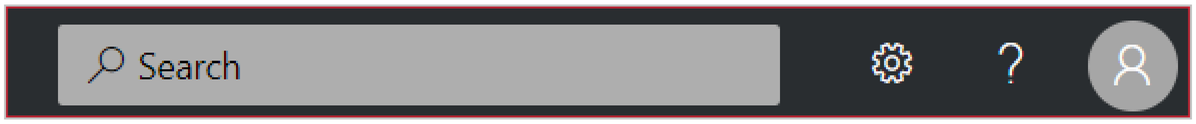
Use the left navigation menu

Use the left navigation menu to access the different areas of the application. You can expand or collapse the navigation bar by selecting **<** or **>**:

Menu	Description
 Dashboard Device Explorer Device Sets Analytics Jobs Device Templates Continuous Data Export Administration	• The **Dashboard** button displays your application dashboard. As a builder, you can customize the dashboard for your operators. Users can also create their own dashboards. • The **Device Explorer** button lists the simulated and real devices associated with each device template in the application. As an operator, you use the **Device Explorer** to manage your connected devices. • The **Device Sets** button enables you to view and create device sets. As an operator, you can create device sets as a logical collection of devices specified by a query. • The **Analytics** button shows analytics derived from device telemetry for devices and device sets. As an operator, you can create custom views on top of device data to derive insights from your application. • The **Jobs** button enables bulk device management by having you create and run jobs to perform updates at scale. • The **Device Templates** button shows the tools a builder uses to create and manage device templates. • The **Continuous Data Export** button an administrator to configure a continuous export to other Azure services such as storage and queues. • The **Administration** button shows the application administration pages where an administrator can manage application settings, users, and roles.

Search, help, and support

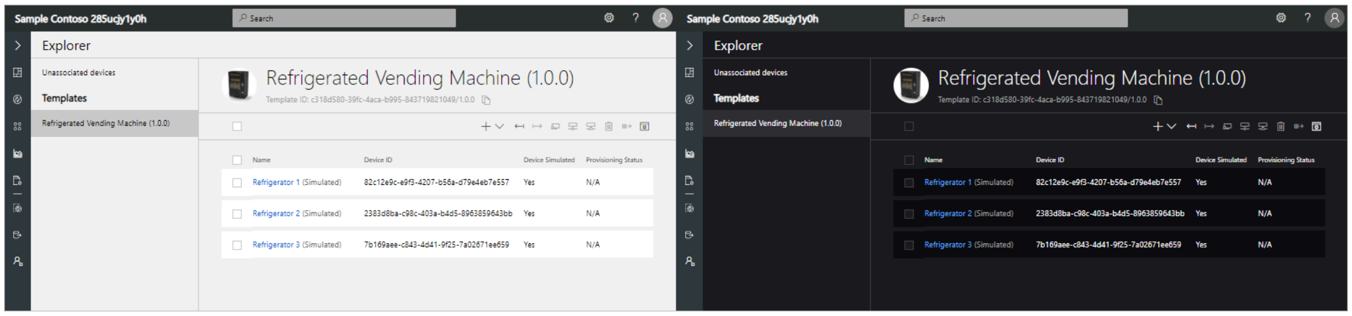
The top menu appears on every page:



- To search for device templates and devices, enter a **Search** value.
- To change the UI language or theme, choose the **Settings** icon.
- To sign out of the application, choose the **Account** icon.
- To get help and support, choose the **Help** drop-down for a list of resources. In a trial

application, the support resources include access to live chat.

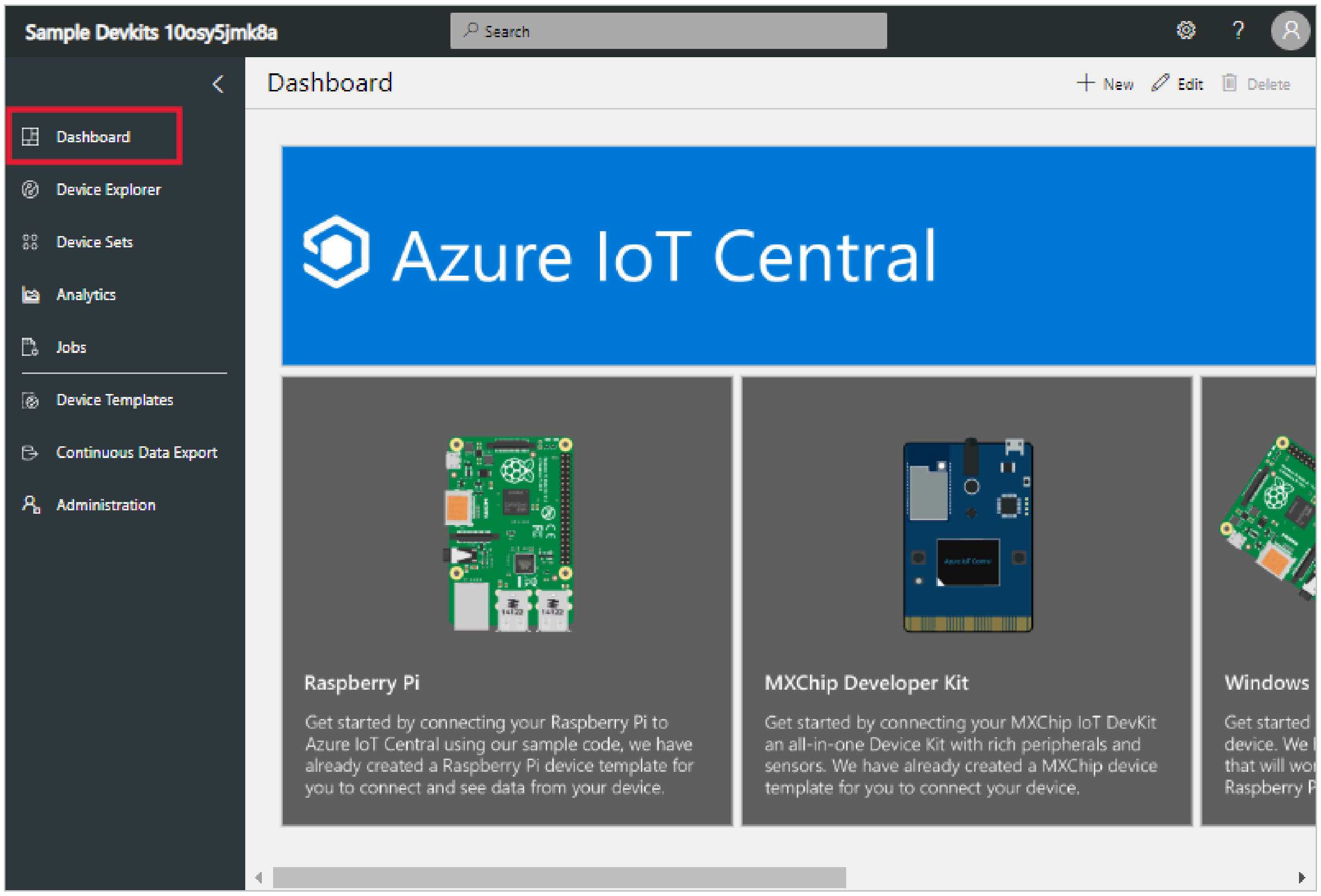
You can choose between a light theme or a dark theme for the UI:



Sample Contoso 285ucjy1y0h
Search
Explorer
Unassociated devices
Templates
Refrigerated Vending Machine (1.0.0)
Refrigerated Vending Machine (1.0.0)
Template ID: c318d580-39fc-4aca-b995-843719821049/1.0.0

Name	Device ID	Device Simulated	Provisioning Status
Refrigerator 1 (Simulated)	82c12e9c-e9f3-4207-b56a-d79e4eb7e557	Yes	N/A
Refrigerator 2 (Simulated)	2383d8ba-c98c-403a-b4d5-8963859643bb	Yes	N/A
Refrigerator 3 (Simulated)	7b169aee-c843-4d41-9f25-7a02671ee659	Yes	N/A

Dashboard



Raspberry Pi

Get started by connecting your Raspberry Pi to Azure IoT Central using our sample code, we have already created a Raspberry Pi device template for you to connect and see data from your device.

MXChip Developer Kit

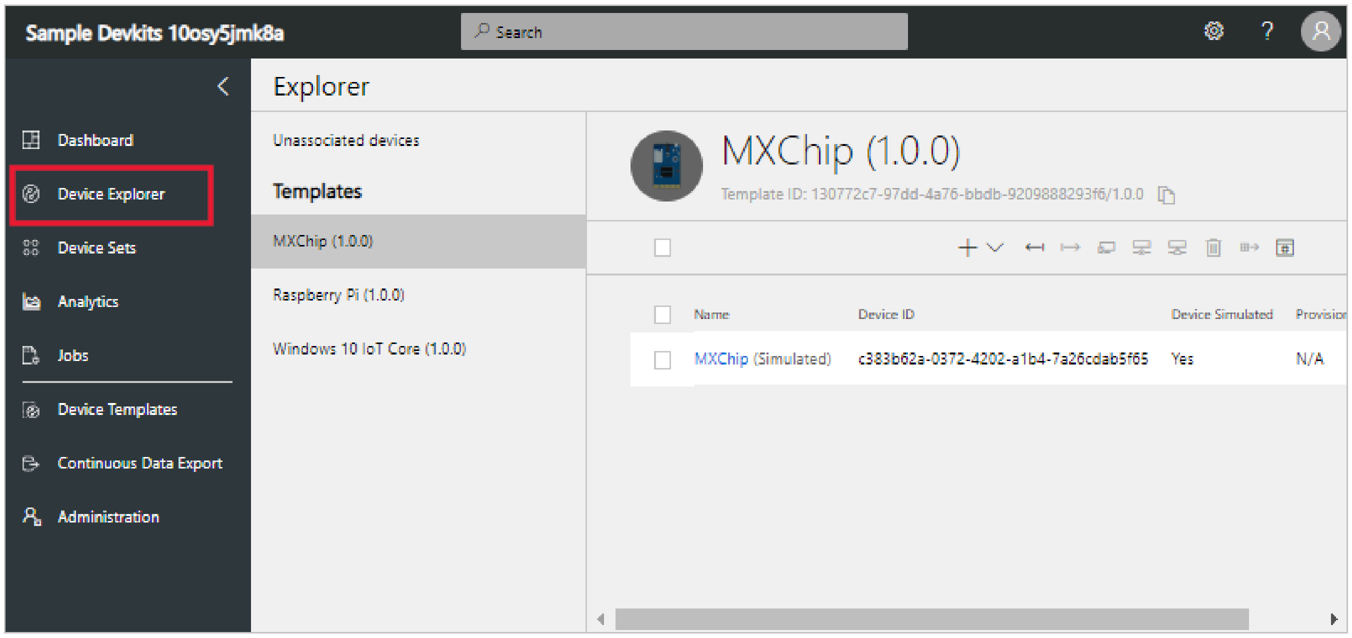
Get started by connecting your MXChip IoT DevKit an all-in-one Device Kit with rich peripherals and sensors. We have already created a MXChip device template for you to connect your device.

Windows

Get started
device. We
that will wo
Raspberry P

The dashboard is the first page you see when you sign in to your Azure IoT Central application. As a builder, you can customize the application dashboard for other users by adding tiles. To learn more, see the Customize the Azure IoT Central operator's view tutorial. Users can also create their own personal dashboards.

Device explorer

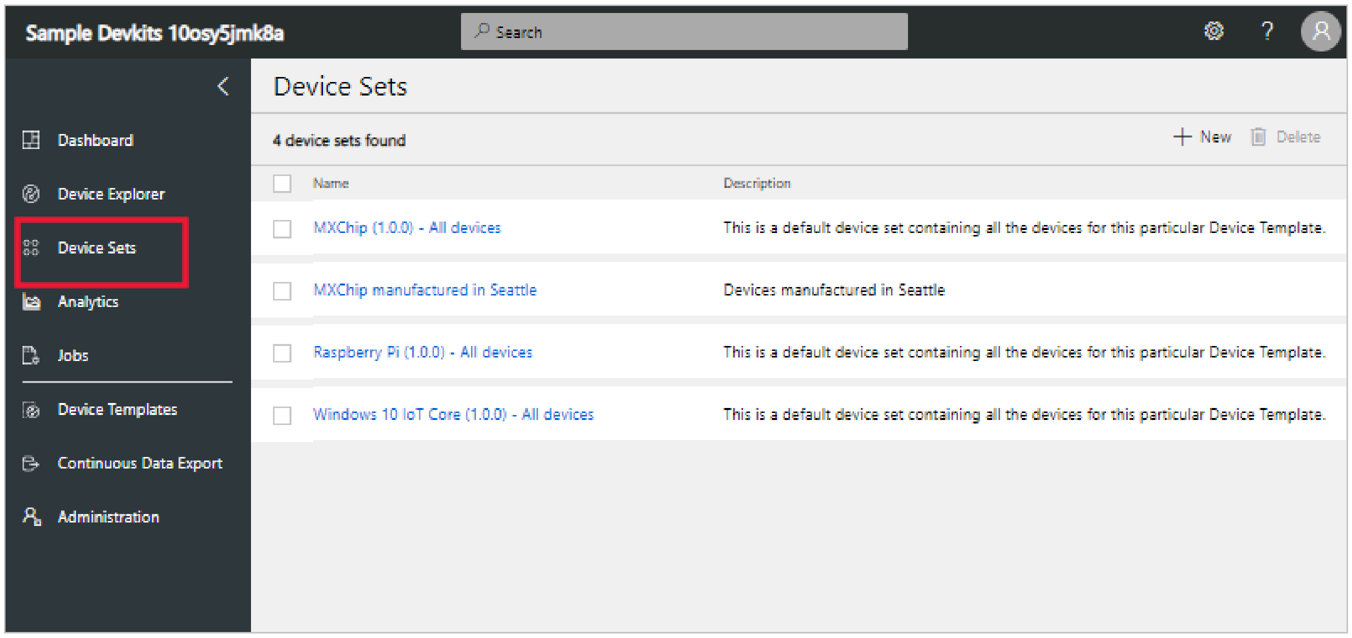


Name	Device ID	Device Simulated	Provisior
MXChip (Simulated)	c383b62a-0372-4202-a1b4-7a26cdab5f65	Yes	N/A

The explorer page shows the *devices* in your Azure IoT Central application grouped by *device template*.

- A device template defines a type of device that can connect to your application. To learn more, see the Define a new device type in your Azure IoT Central application.
- A device represents either a real or simulated device in your application. To learn more, see the Add a new device to your Azure IoT Central application.

Device sets


Sample Devkits 10osy5jmk8a

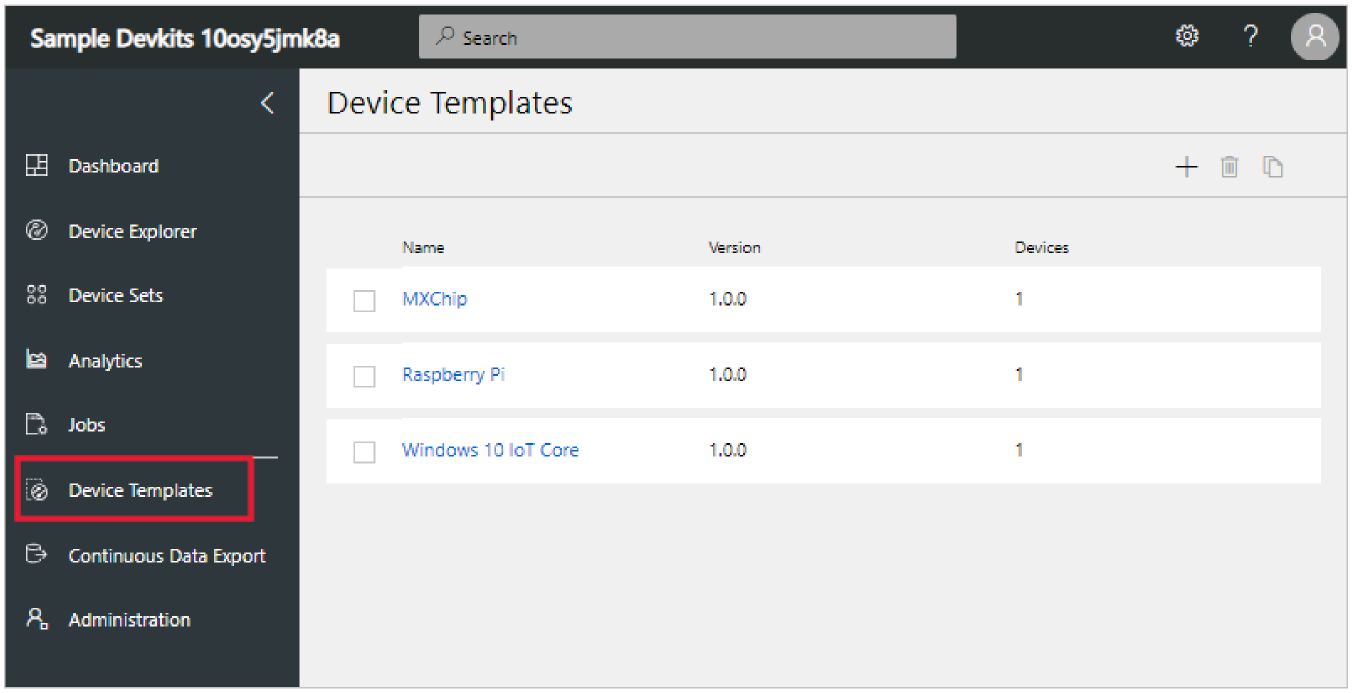
Device Sets

4 device sets found

Name	Description
MXChip (1.0.0) - All devices	This is a default device set containing all the devices for this particular Device Template.
MXChip manufactured in Seattle	Devices manufactured in Seattle
Raspberry Pi (1.0.0) - All devices	This is a default device set containing all the devices for this particular Device Template.
Windows 10 IoT Core (1.0.0) - All devices	This is a default device set containing all the devices for this particular Device Template.

The *device sets* page shows device sets created by the builder. A device set is a collection of related devices. A builder defines a query to identify the devices that are included in a device set. You use device sets when you customize the analytics in your application. To learn more, see the Use device sets in your Azure IoT Central application article.

Device Templates

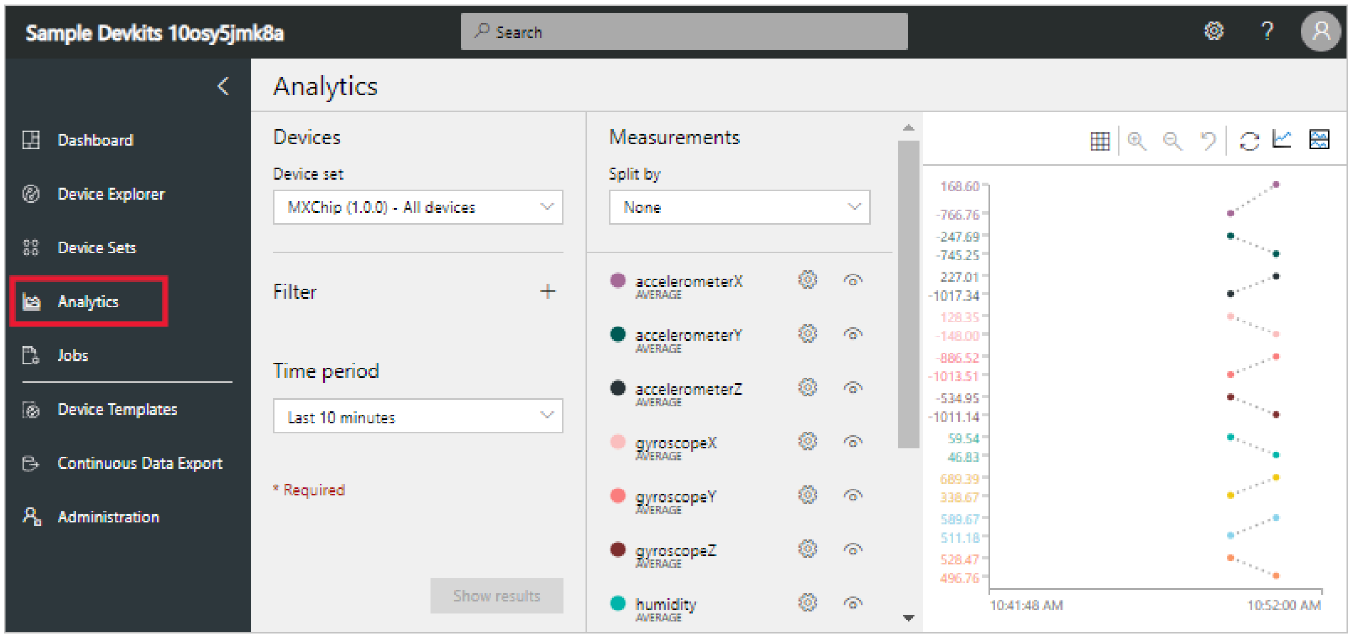

Sample Devkits 10osy5jmk8a

Device Templates

Name	Version	Devices
MXChip	1.0.0	1
Raspberry Pi	1.0.0	1
Windows 10 IoT Core	1.0.0	1

The device templates page is where a builder creates and manages the device templates in the application. To learn more, see the Define a new device type in your Azure IoT Central application tutorial.

Analytics



Device set: MXChip (1.0.0) - All devices

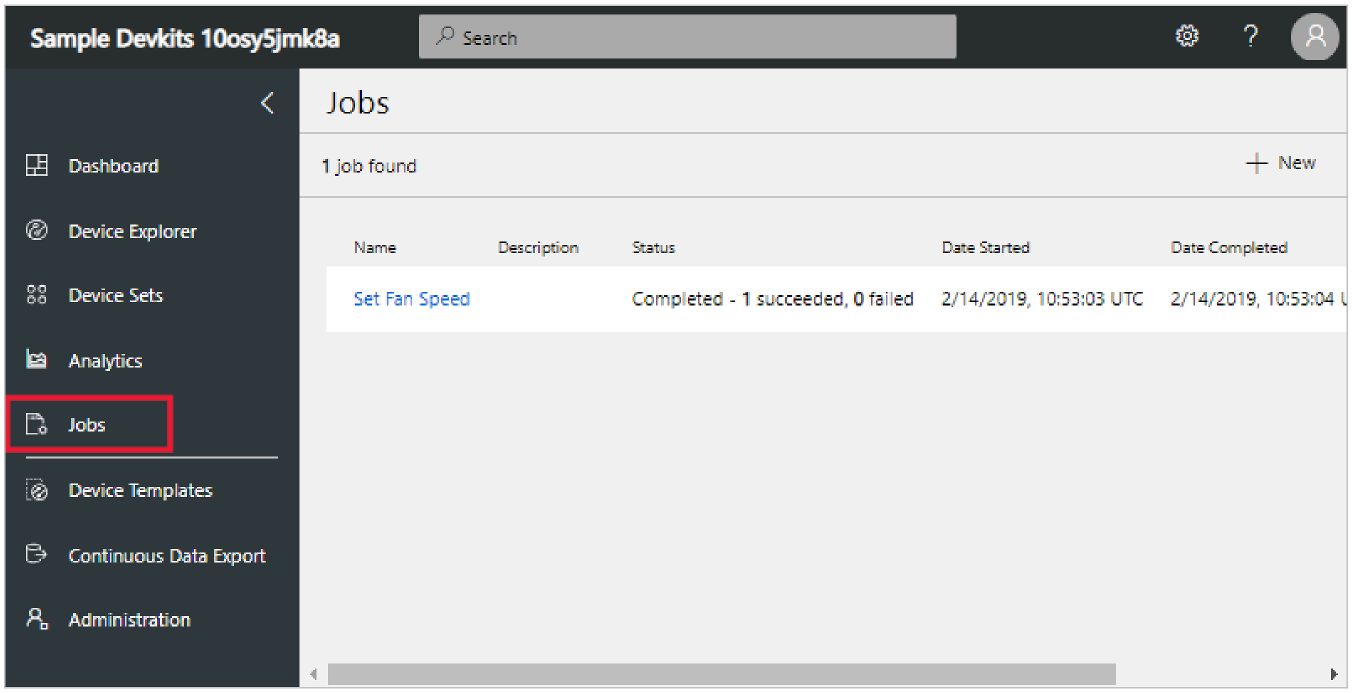
Split by: None

Time period: Last 10 minutes

* Required

The analytics page shows charts that help you understand how the devices connected to your application are behaving. An operator uses this page to monitor and investigate issues with connected devices. The builder can define the charts shown on this page. To learn more, see the Create custom analytics for your Azure IoT Central application article.

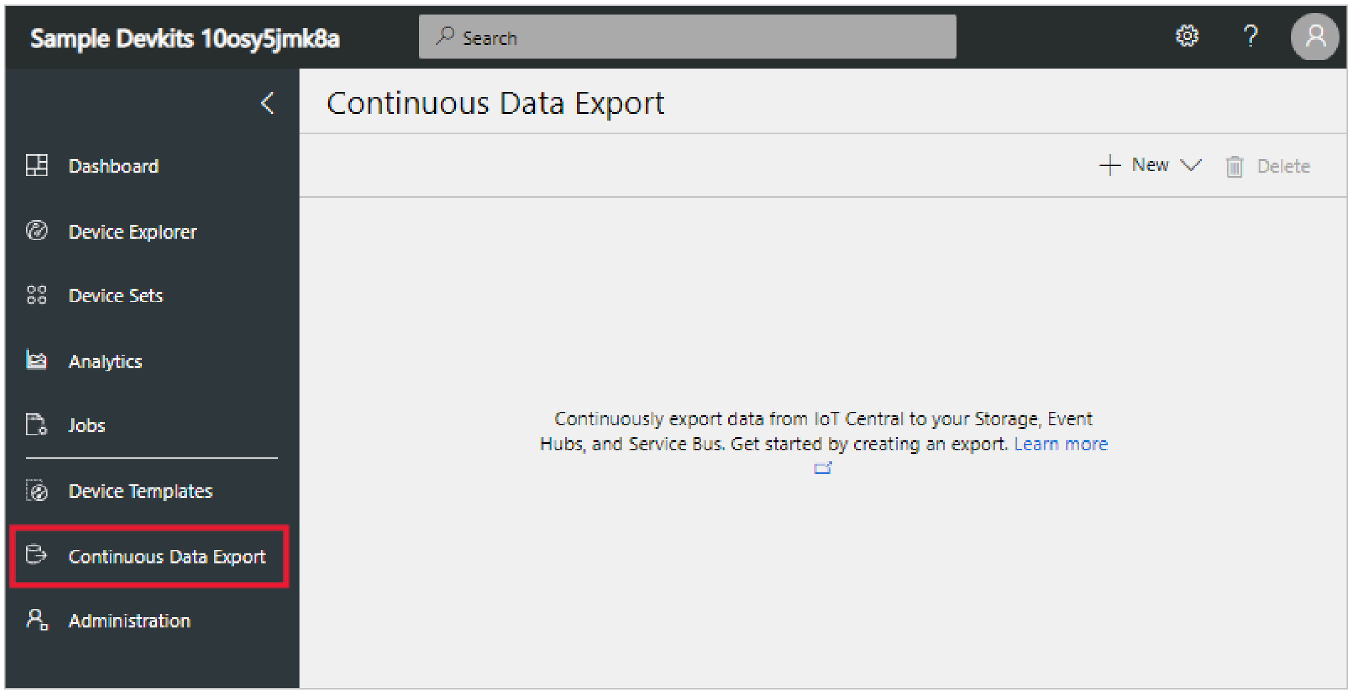
Jobs



Name	Description	Status	Date Started	Date Completed
Set Fan Speed		Completed - 1 succeeded, 0 failed	2/14/2019, 10:53:03 UTC	2/14/2019, 10:53:04 U

The jobs page allows you to perform bulk device management operations onto your devices. The builder uses this page to update device properties, settings, and commands. To learn more, see the Run a job article.

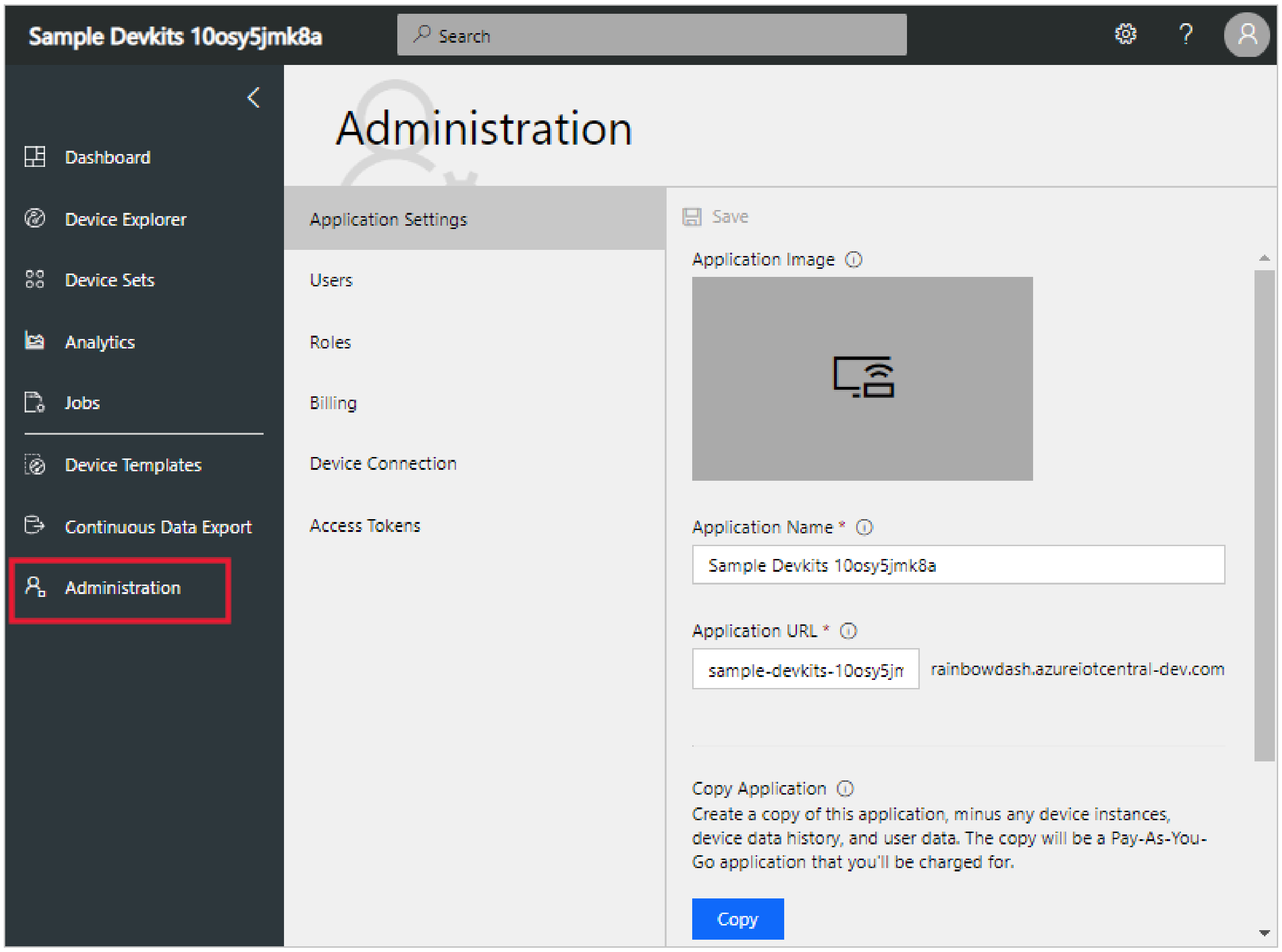
Continuous Data Export



Continuously export data from IoT Central to your Storage, Event Hubs, and Service Bus. Get started by creating an export. Learn more

The continuous data export page is where an administrator defines how to export data, such as telemetry, from the application. Other services can store the exported data or use it for analysis. To learn more, see the Export your data in Azure IoT Central article.

Administration



Application Name: Sample Devkits 10osy5jmk8a

Application URL: sample-devkits-10osy5jn rainbowdash.azureiotcentral-dev.com

Create a copy of this application, minus any device instances, device data history, and user data. The copy will be a Pay-As-You-Go application that you'll be charged for.

The administration page contains links to the tools an administrator uses such as defining users and roles in the application. To learn more, see the Administer your Azure IoT Central application article.