



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
Hardware	Raspberry Pi 4 . 4GB model required for 20 Users. Raspberry Pi Sense HAT , Optional: Raspberry Pi case , active cooling fan
USB3 SSD Drive	To support up to 20 users per Raspberry Pi you need a fast USB3 SSD Drive to run Raspbian Buster Linux on. A 120 USB3 SSD drive will be more than sufficient. These are readily available from online stores.
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 a Python Internet of Things (IoT) application with [Visual Studio Code](#). Run the application in a Docker Container on a Raspberry Pi, read temperature, humidity, and air pressure telemetry from a sensor, and finally debug the application running in the Docker Container.



References

- [Visual Studio Code](#)
- [Azure IoT Central](#)
- [Installing Docker on Raspberry Pi Buster](#)
- [Understanding Docker in 12 Minutes](#)

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](#)

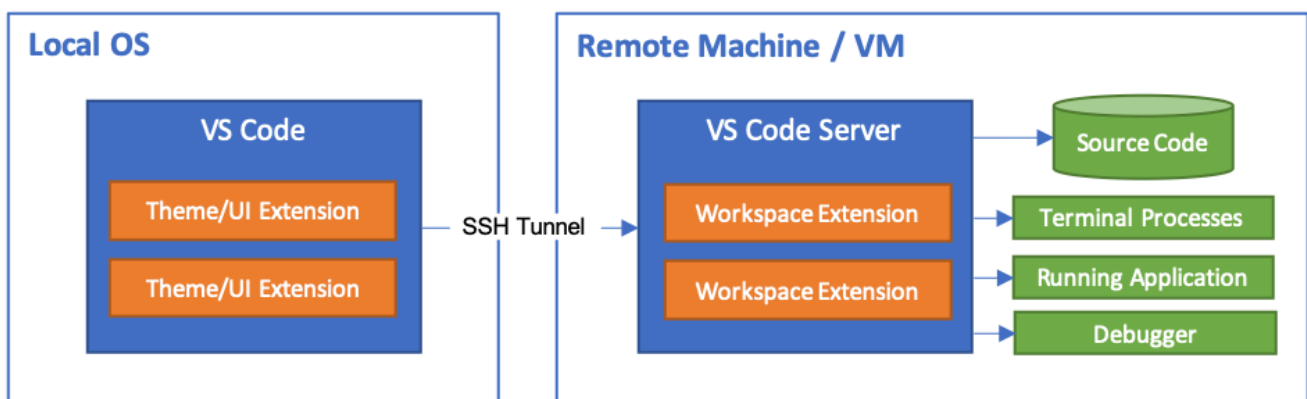
For information on contributing or submitting issues see the [Visual Studio GitHub Repository](#).

Visual Studio Code documentation is also Open Source, and you can contribute or submit issues from the [Visual Studio Documentation GitHub Repository](#).

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.



Raspberry Pi Hardware

If you are attending a workshop, then you can use a shared network-connected Raspberry Pi. 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**.

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](#)

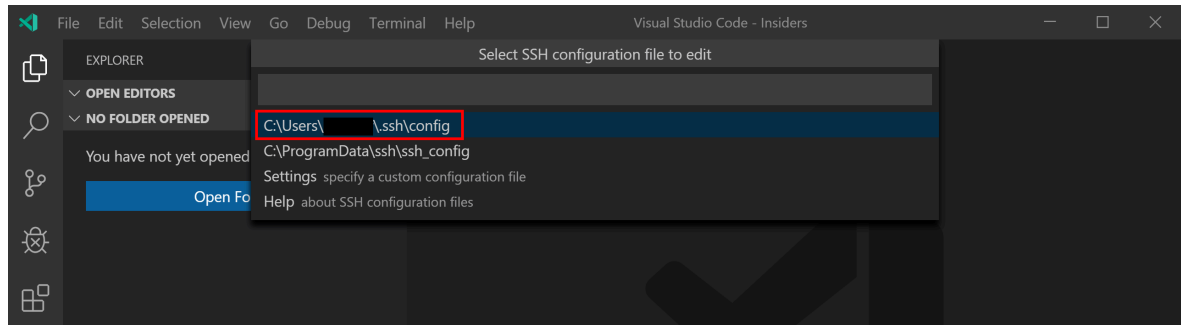
Trouble Shooting SSH Client Installation

- [Remote Development using SSH](#)
- [Installing a supported SSH client](#)

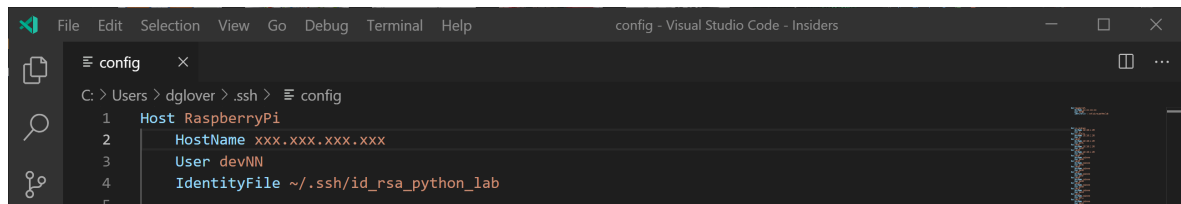
Configure Visual Studio Code Remote SSH Development

1. Start Visual Studio Code Insiders Edition

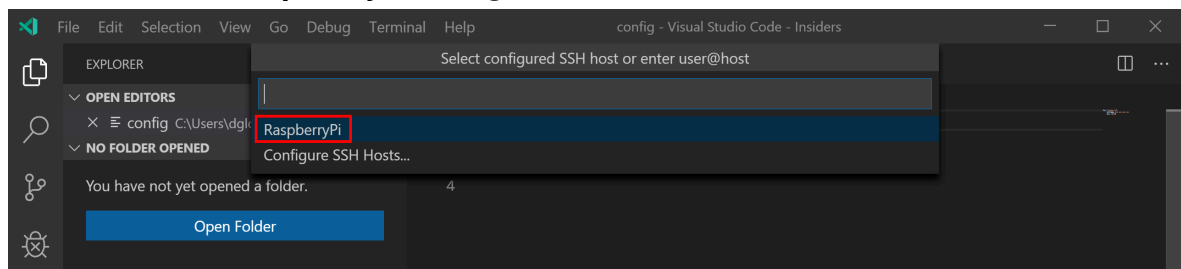
2. Press F1 to open the Command Palette, type **ssh config** and select **Remote-SSH: Open Configuration**
3. Select the user `.ssh config` file



4. Set the SSH connection configuration as follows:
 - **Host:** Set to **RaspberryPi**
 - **HostName:** The Raspberry Pi IP Address
 - **User:** Your **login name**
 - **IdentityFile:** Set to `~/.ssh/id_rsa_python_lab`.
 - Save these changes (Ctrl+S).



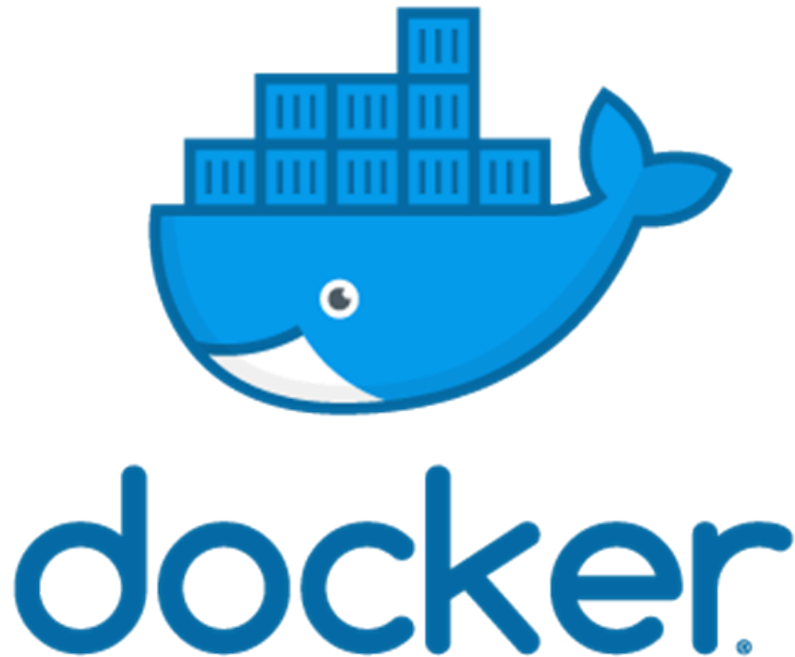
5. Press **F1** to open the Command Palette, type **ssh connect** and select **Remote-SSH: Connect to Host**
6. Select the host **RaspberryPi** configuration



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

Introduction to Docker

[Jake Wright's Docker in 12 Minutes](#) is a great introduction to Docker.



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.



Create a New IoT Central Application

1. Open the [Azure IoT Central](#) in a new browser tab, then click **Getting started**.
2. Next, you will 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.



Sign in

to continue to Azure IoT Central

Email, phone, or Skype

[Can't access your account?](#)

No account? [Create one!](#)

Next

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.

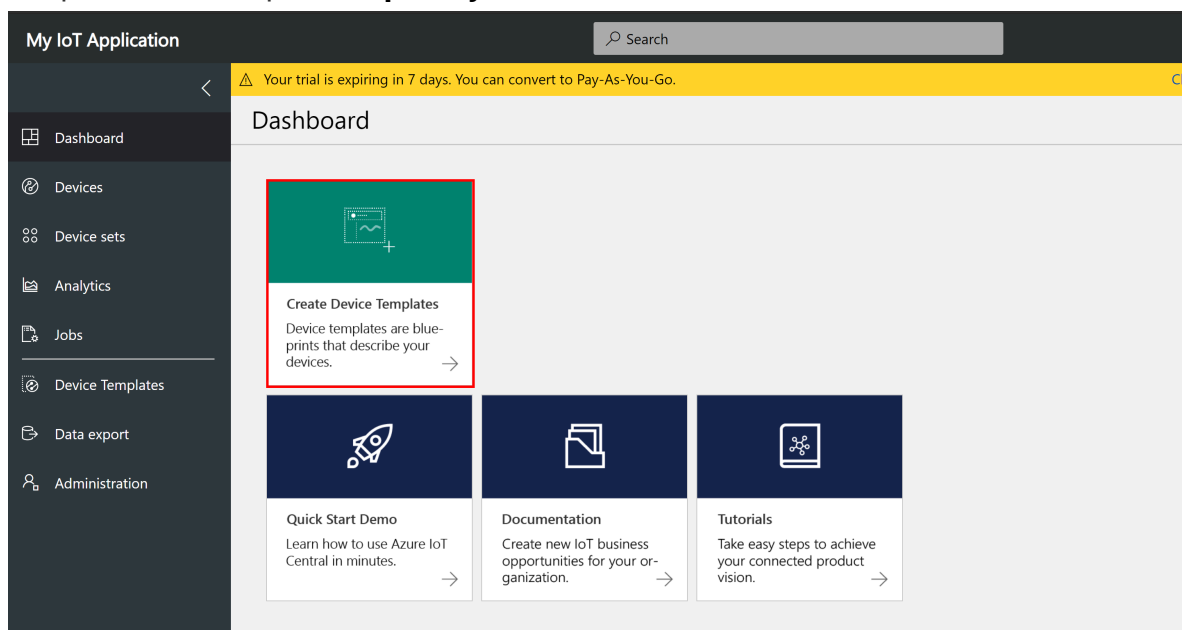
Choose a payment plan

<input checked="" type="radio"/> Trial	<input type="radio"/> Pay-As-You-Go
Free trial for 7 days. No subscription required.	Price is based on the number of devices you use. Free for the first 5 devices. Subscription required. Learn more

Select an application template

<input type="radio"/> Sample Contoso	<input type="radio"/> Sample Devkits	<input checked="" type="radio"/> Custom application
Get started with a predefined application for a connected device.	Want to connect a Raspberry PI or MXChip IoT DevKit? Start with this predefined app and get them connected in minutes.	Start with a blank template and define your application from scratch.

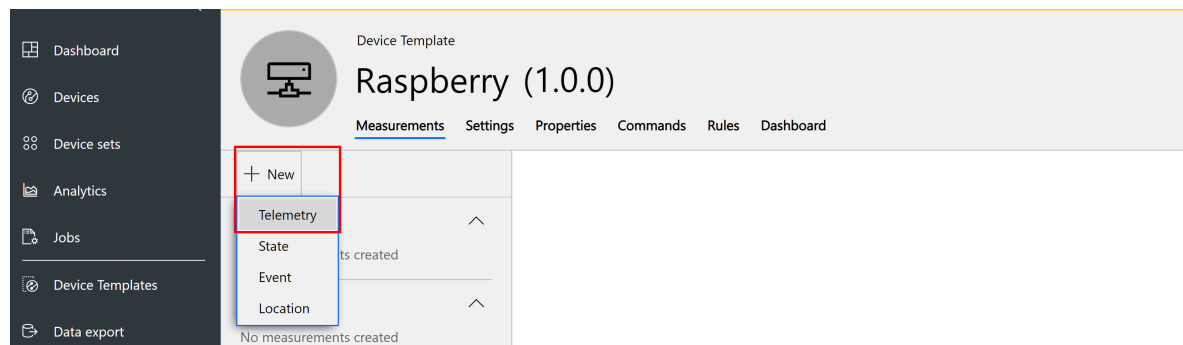
4. Click **Create Device Templates**, then select the **Custom** template, name your template, for example, **Raspberry**. Then click Create



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.



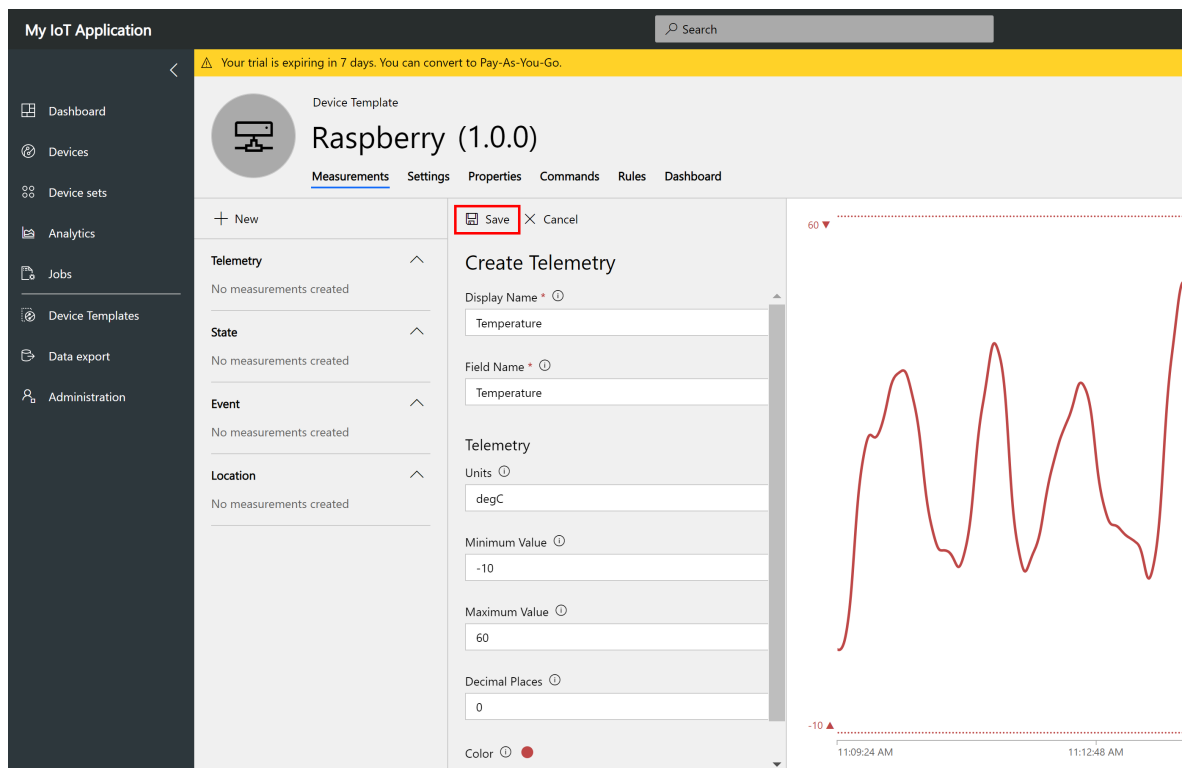
Use the information in the following table to set up three telemetry measurements.

The field name is case-sensitive.

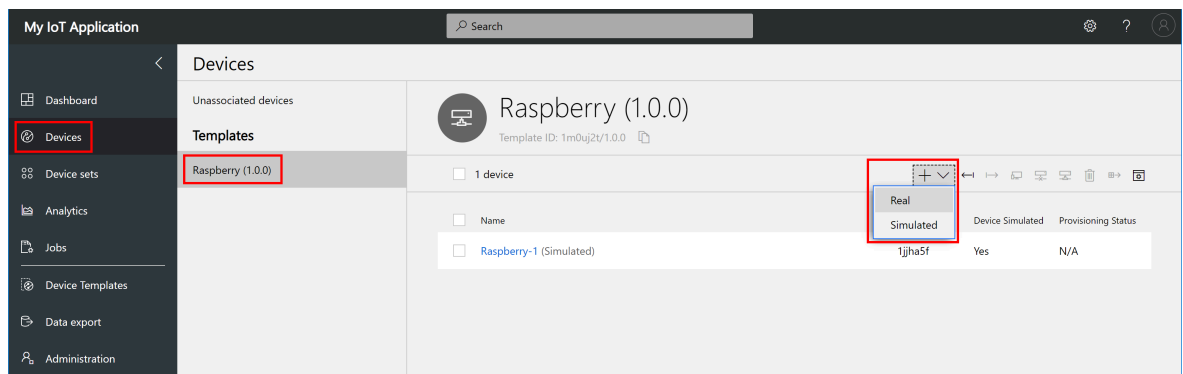
You **must** click **Save** after each measurement is defined.

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 * ⓘ

raspberrypi-01

↺

📄

Device Name ⓘ

Raspberry - raspberrypi-01

↺

📄

Create

Cancel

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.

My IoT Application

Search

⚙️ ? 👤

Dashboard

Devices

Device sets

Analytics

Jobs

Device Templates

Device

Raspberry - raspberrypi-01

Block Connect Delete

Measurements Settings Properties Commands Rules Dashboard

Status: Registered

Use the measurements to monitor your device data.

Telemetry

Temperature AVERAGE

State

View: list table user

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+jA23PeMWEQUP35Q=

Secondary Key ⓘ
EQ3iQjiy14kvKV3Gdjx5a/HgbNQ=

Close

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	01ne333333333333
Device Id	my-device
Device Key	UzztjCvwxQmKaszy5RAn/j+bSEfEjBW7w3V145lp/c=

Get Connection String

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

Configure the Python Application

1. Switch back to Visual Studio Code. Open the **env-file** (environment file). This file contains environment variables that will be passed to 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)

Build and Run the Docker Image

Press **F5** to start debugging the Python application. The process will first build and then start the Docker Container. When the Docker Container has started the Visual Studio Code Debugger will attach to the running application.

There are two configuration files found in the .vscode folder that are responsible for running and debugging the Python application. You can find more detail the [Debugger Configuration](#) appendix.

Set a Visual Studio Debugger Breakpoint

1. From **Explorer** on the Visual Studio Code activity bar, open the **app.py** file
2. Set a breakpoint at line 66, **temperature, pressure, humidity, timestamp = mysensor.measure()** in the **publish** function.

- You can set a breakpoint 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:

```
62 def publish():
63     msgId = 1
64     while True:
65         try:
66             temperature, pressure, humidity, timestamp, cpu_temperature = mysensor.measure()
67
68             data = {
69                 "Geo": 'Sydney, AU',
70                 "Humidity": humidity,
71                 "Pressure": pressure,
72                 "Temperature": temperature,
73                 "CpuTemperature": cpu_temperature,
74                 "Epoch": timestamp,
75                 "Id": msgId
76             }
```

Debug actions

Once a debug session starts, the **Debug toolbar** will appear at the top of the editor window.



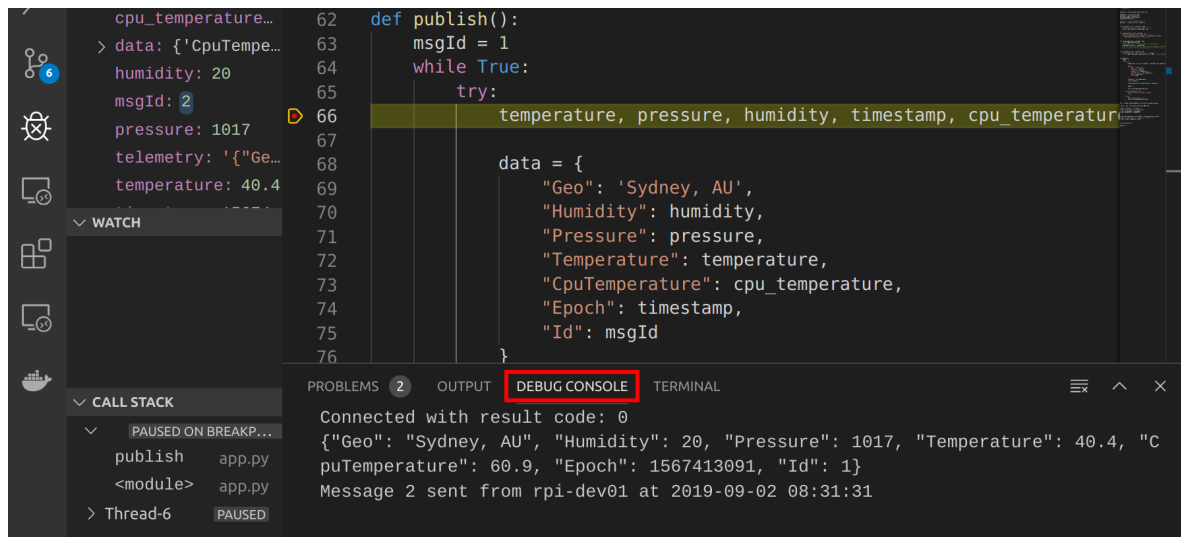
The debugger toolbar (shown above) will appear in Visual Studio Code. It has the following options:

1. Pause (or Continue, F5),
2. Step Over (F10)
3. Step Into (F11),
4. Step Out (Shift+F11),
5. Restart (Ctrl+Shift+F5),
6. and Stop (Shift+F5).

Step through the Python code

1. Press **F10**, or from the Debugger Toolbar, click **Step Over** until you are past the **print(telemetry)** line of code.
2. Explore the **Variable Window** (Ctrl+Shift+Y). Try changing variable values.

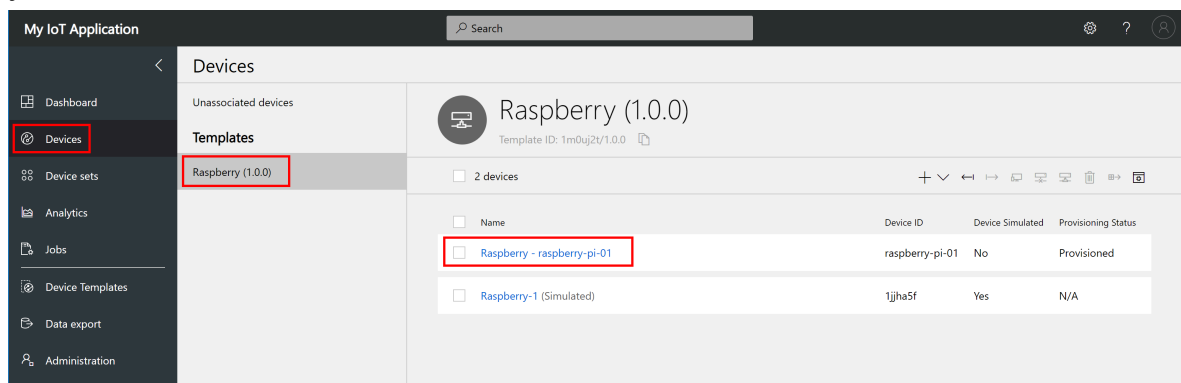
3. Explore the **Debug Console**. You will see sensor telemetry and the results of sending the telemetry to Azure IoT Central.



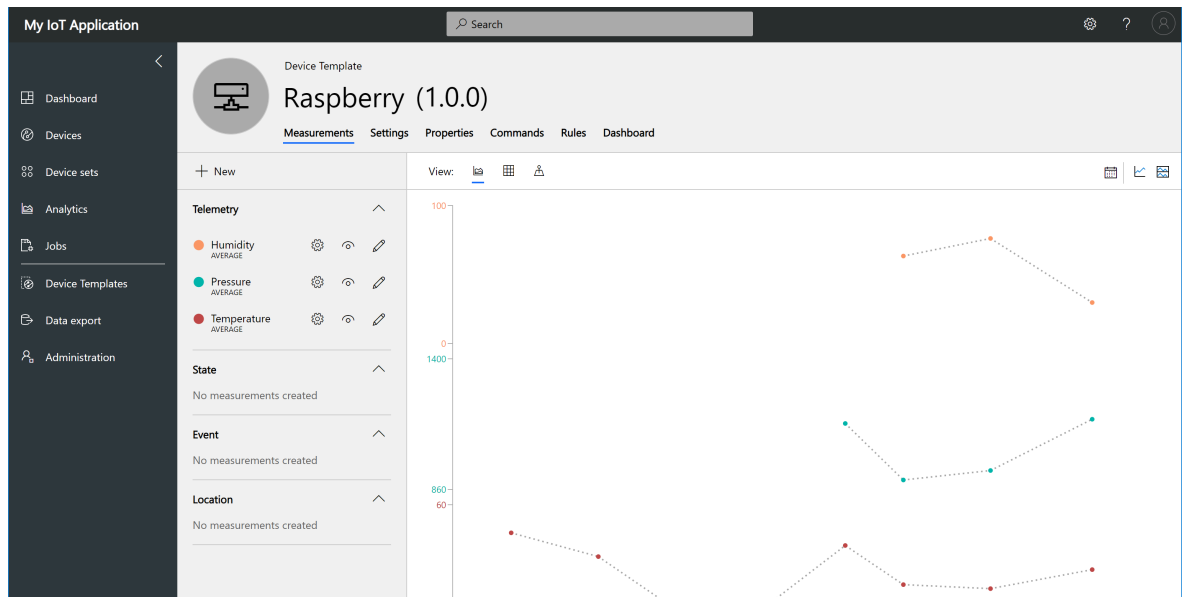
4. From the **Debug** Menu -> **Disable All Breakpoints**
5. Press **F5** or from the Debugger Toolbar, click **Continue** so the Python application runs and streams telemetry to **Azure IoT Central**.

Exploring Device Telemetry in Azure IoT Central

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



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



Finished

Complete. Congratulations

Appendix

Debugger Configuration

There are two files (launch.json and tasks.json) found in the .vscode folder that are responsible for the running and debugging the application.

Launch Configuration

Creating a launch configuration file is useful as it allows you to configure and save debugging setup details.

launch.json

```

{
  // 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": "Attach Debugger",
      "preLaunchTask": "start-docker",
      "postDebugTask": "stop-docker",
      "type": "python",
      "request": "attach",
      "pathMappings": [
        {
          "localRoot": "${workspaceRoot}/app",
          "remoteRoot": "/app"
        }
      ],
      "port": "${env:LAB_PORT}",
      "host": "localhost"
    },
    {
      "name": "Stop Container",
      "preLaunchTask": "stop-docker",
      "type": "python",
      "request": "launch"
    }
  ]
}

```

Tasks Configuration

Tasks integrate external tools to automate build cycle.

tasks.json

```

{
  // See https://go.microsoft.com/fwlink/?LinkId=733558
  // for the documentation about the tasks.json format
  "version": "2.0.0",
  "tasks": [
    {
      "label": "start-docker",
      "type": "shell",
      "command": "sh",
      "args": [
        "-c",
        "\"docker build -t $USER:latest . ;docker run -d -p $LAB_PORT:3000 -e TELE
        // -d Run container in background and print container ID,
        // -p maps the $LAB_PORT to port 3000 in the container, this port is used
        // -e Environment Variable. The IP Address of the telemetry service.
        // --env-file reads from a file and sets Environment Variables in the Dock
        // --name names the Docker Container
        // --rm removes the container when you stop it
        // Docker run reference https://docs.docker.com/engine/reference/run/
      ],
    },
    {
      "label": "stop-docker",
      "type": "shell",
      "command": "sh",
      "args": [
        "-c",
        "\"docker stop $USER\""
      ]
    }
  ]
}

```

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:

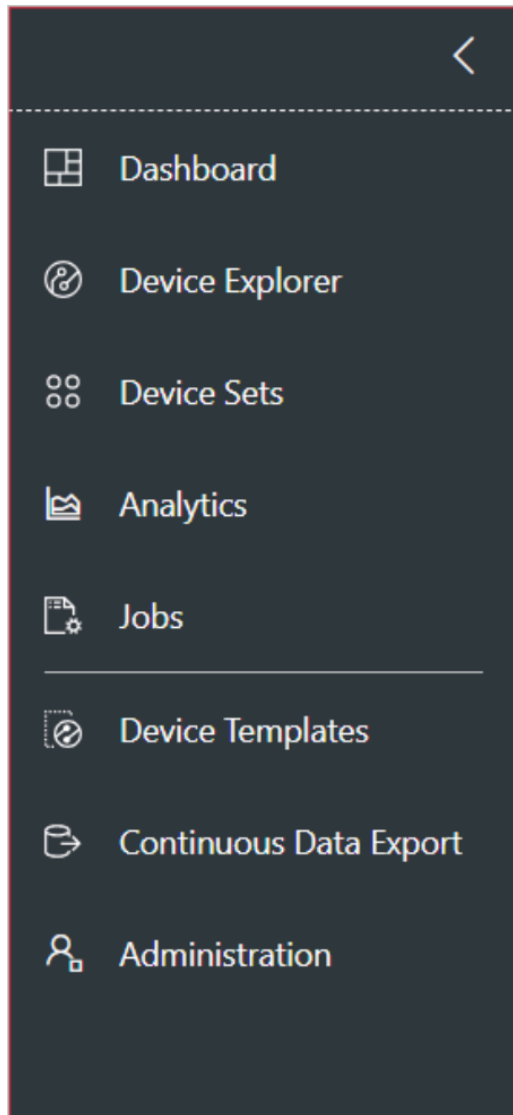
- Monitor your devices.
- Configure your devices.
- Troubleshoot and remediate issues with your devices.
- Provision new devices.

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

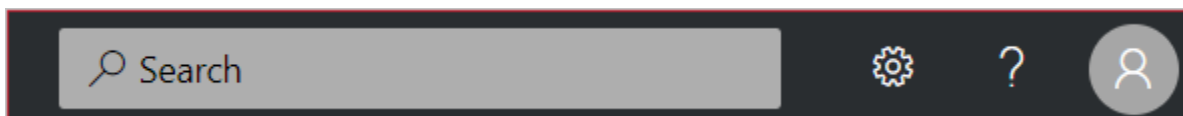
Description



- 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

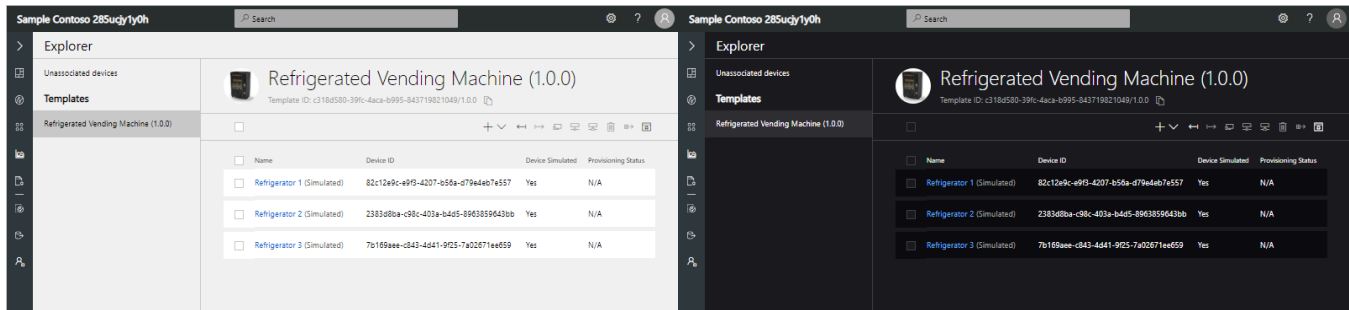
The top menu appears on every page:



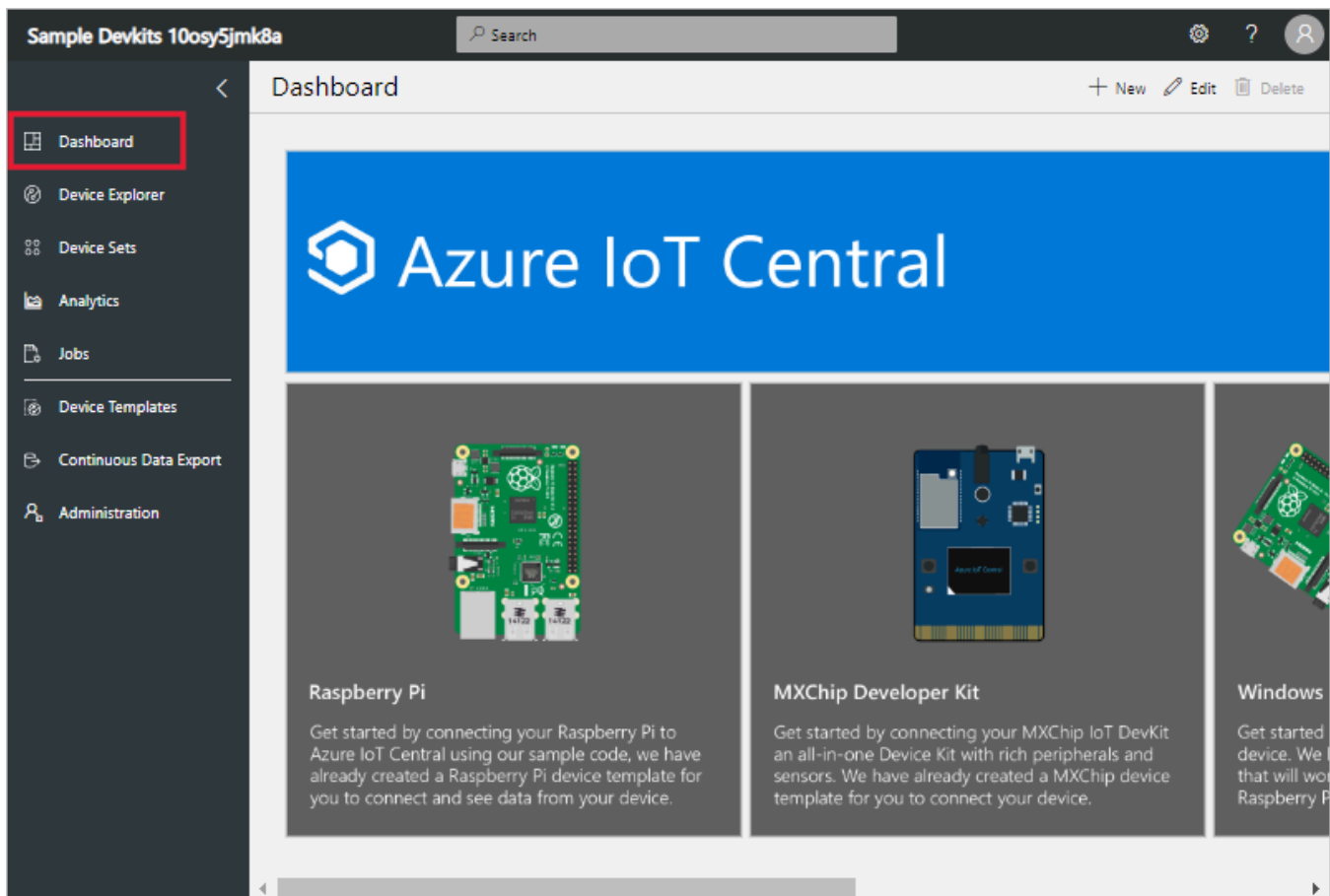
- To search for device templates and devices, enter a **Search** value.
- To change the UI language or theme, choose the **Settings** icon.
- To sign out of the application, choose the **Account** icon.
- To get help and support, choose the **Help** drop-down for a list of resources. In a trial

application, the support resources include access to [live chat](#).

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

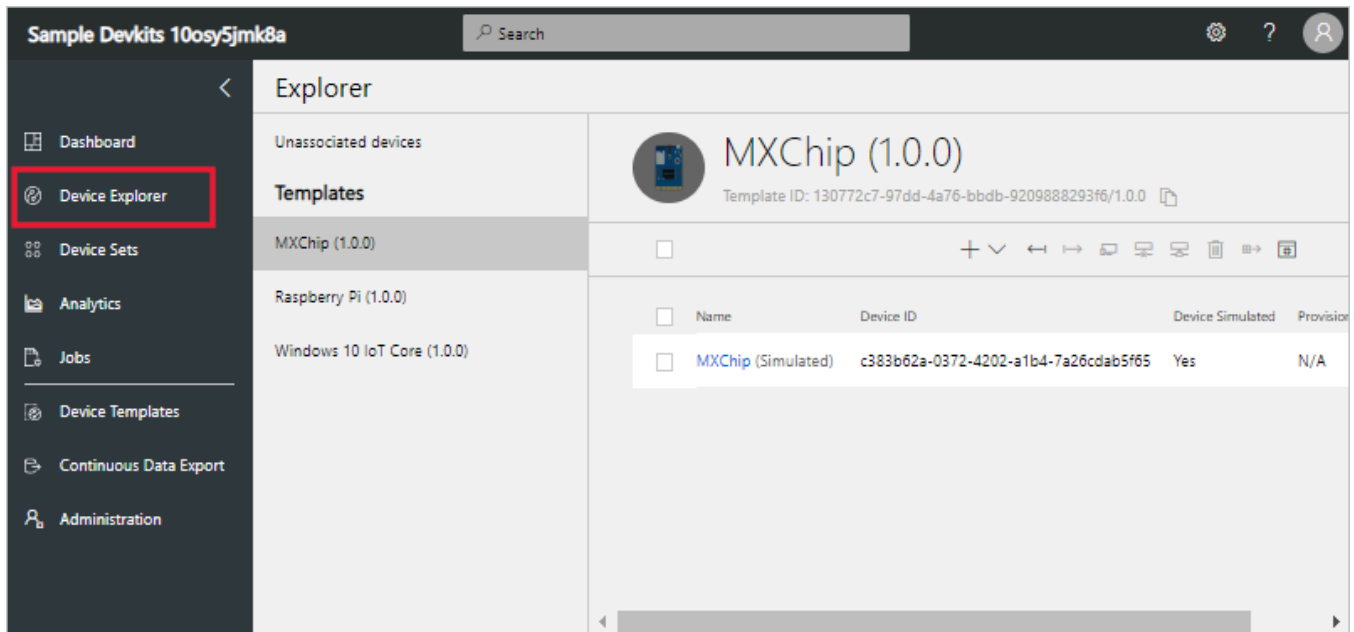


Dashboard



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



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

Search

Device Sets

4 device sets found

+ New Delete

<input type="checkbox"/>	Name	Description
<input type="checkbox"/>	MXChip (1.0.0) - All devices	This is a default device set containing all the devices for this particular Device Template.
<input type="checkbox"/>	MXChip manufactured in Seattle	Devices manufactured in Seattle
<input type="checkbox"/>	Raspberry Pi (1.0.0) - All devices	This is a default device set containing all the devices for this particular Device Template.
<input type="checkbox"/>	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

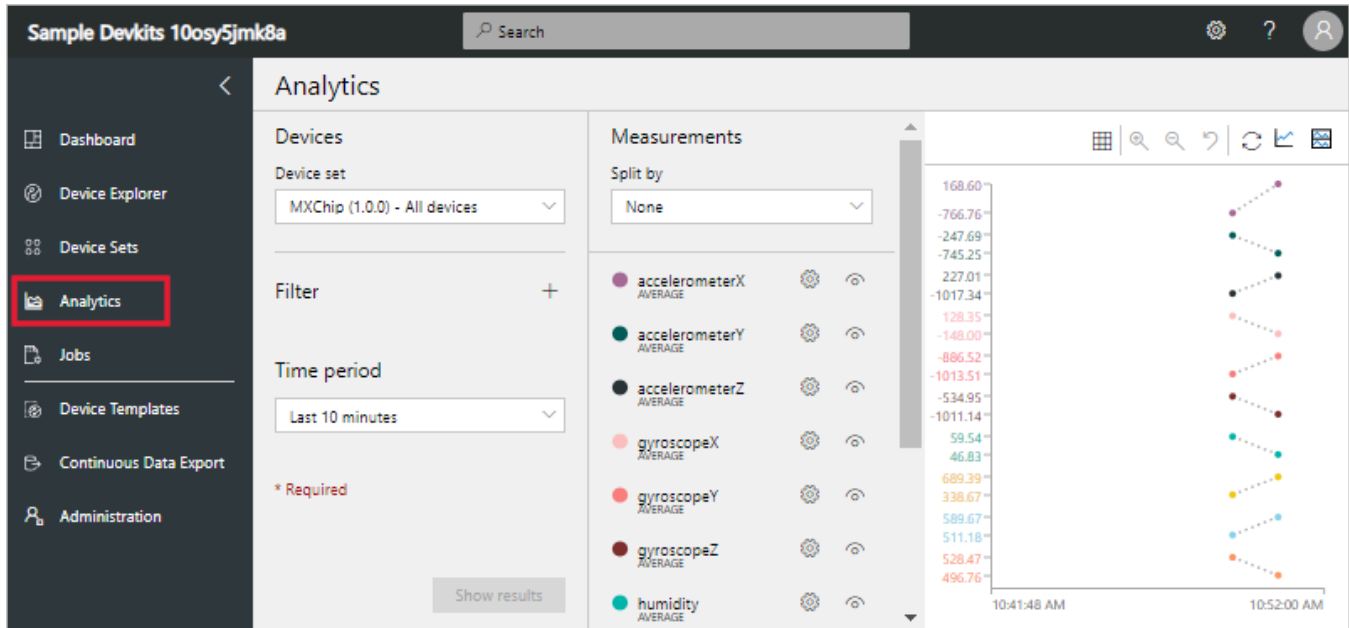
Search

Device Templates

<input type="checkbox"/>	Name	Version	Devices
<input type="checkbox"/>	MXChip	1.0.0	1
<input type="checkbox"/>	Raspberry Pi	1.0.0	1
<input type="checkbox"/>	Windows 10 IoT Core	1.0.0	1

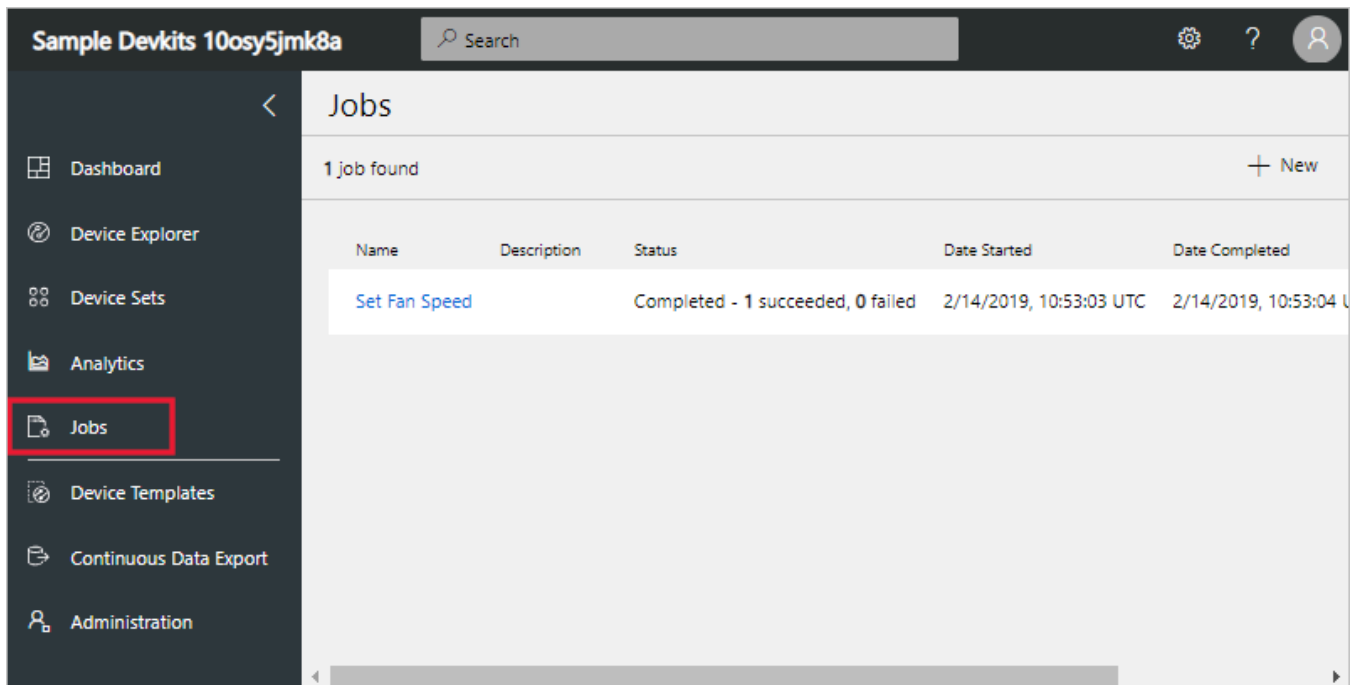
The device templates page is where a builder creates and manages the device templates in the application. To learn more, see the [Define a new device type in your Azure IoT Central application](#) tutorial.

Analytics



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

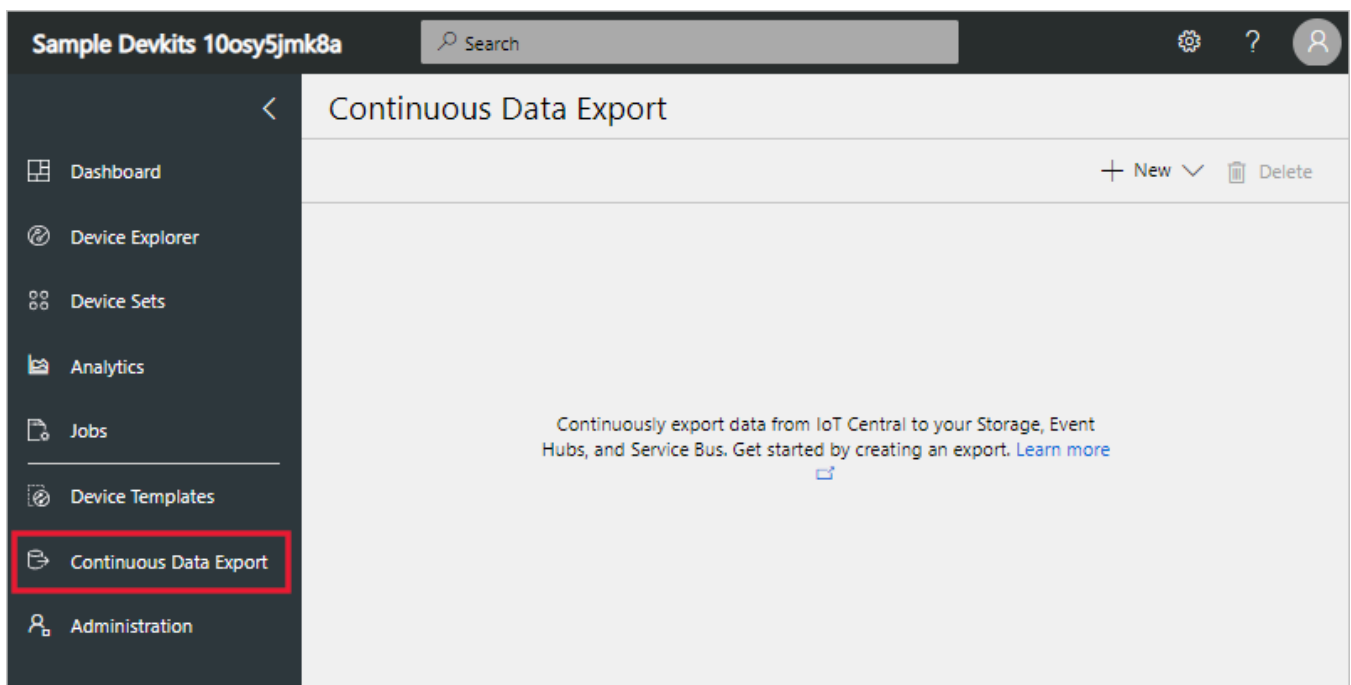


The screenshot shows the 'Jobs' page in the IoT Central interface. The left sidebar contains a navigation menu with items: Dashboard, Device Explorer, Device Sets, Analytics, Jobs (highlighted with a red box), Device Templates, Continuous Data Export, and Administration. The main content area is titled 'Jobs' and shows '1 job found'. A table lists the job details:

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 UTC

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.

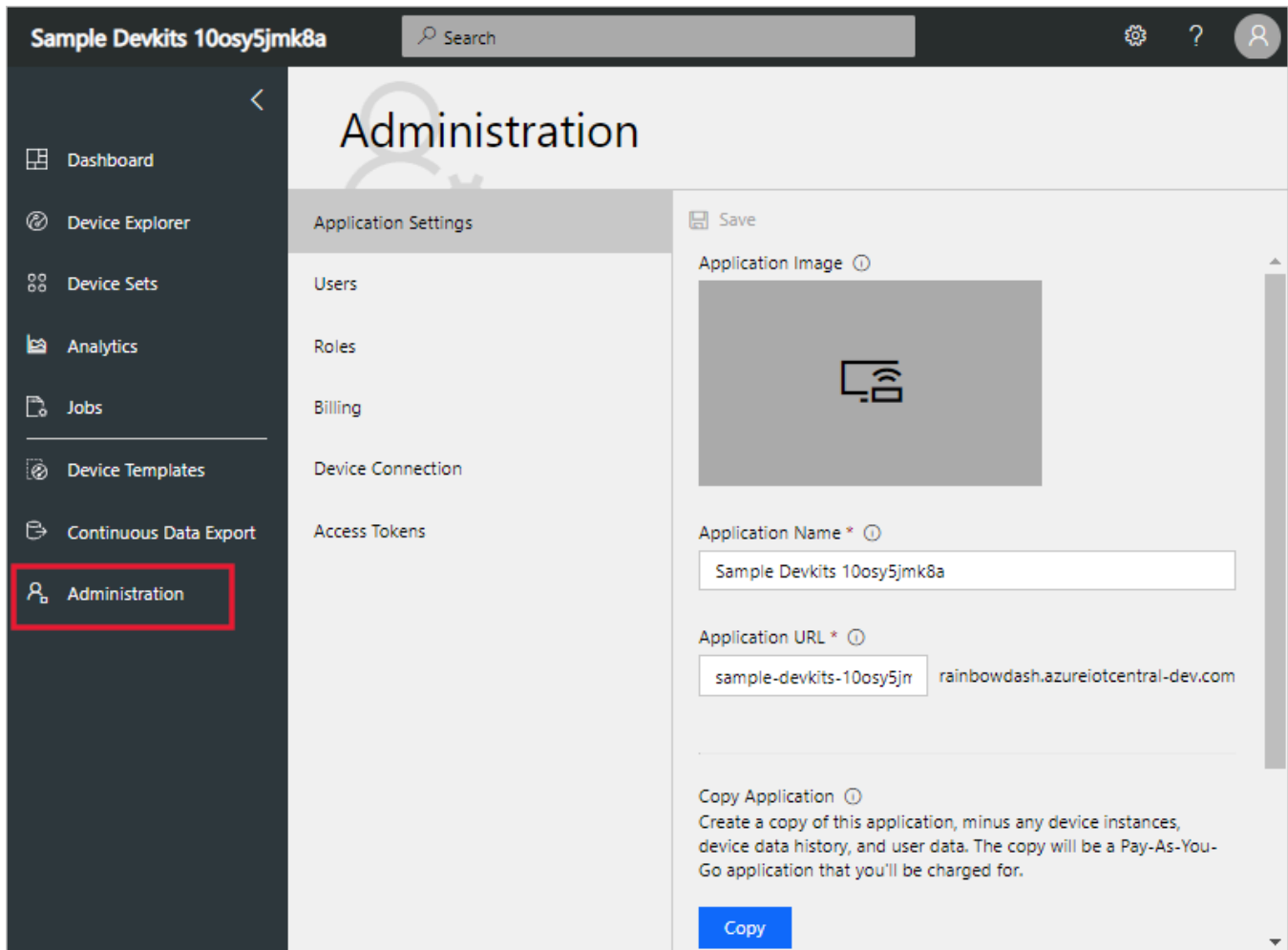
Continuous Data Export



The screenshot shows the 'Continuous Data Export' page in the IoT Central interface. The left sidebar contains a navigation menu with items: Dashboard, Device Explorer, Device Sets, Analytics, Jobs, Device Templates, Continuous Data Export (highlighted with a red box), and Administration. The main content area is titled 'Continuous Data Export' and shows a '+ New' button and a 'Delete' button. Below the buttons, there is a message: '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



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.