



.NET IoT and AI Hack Days - Session 1



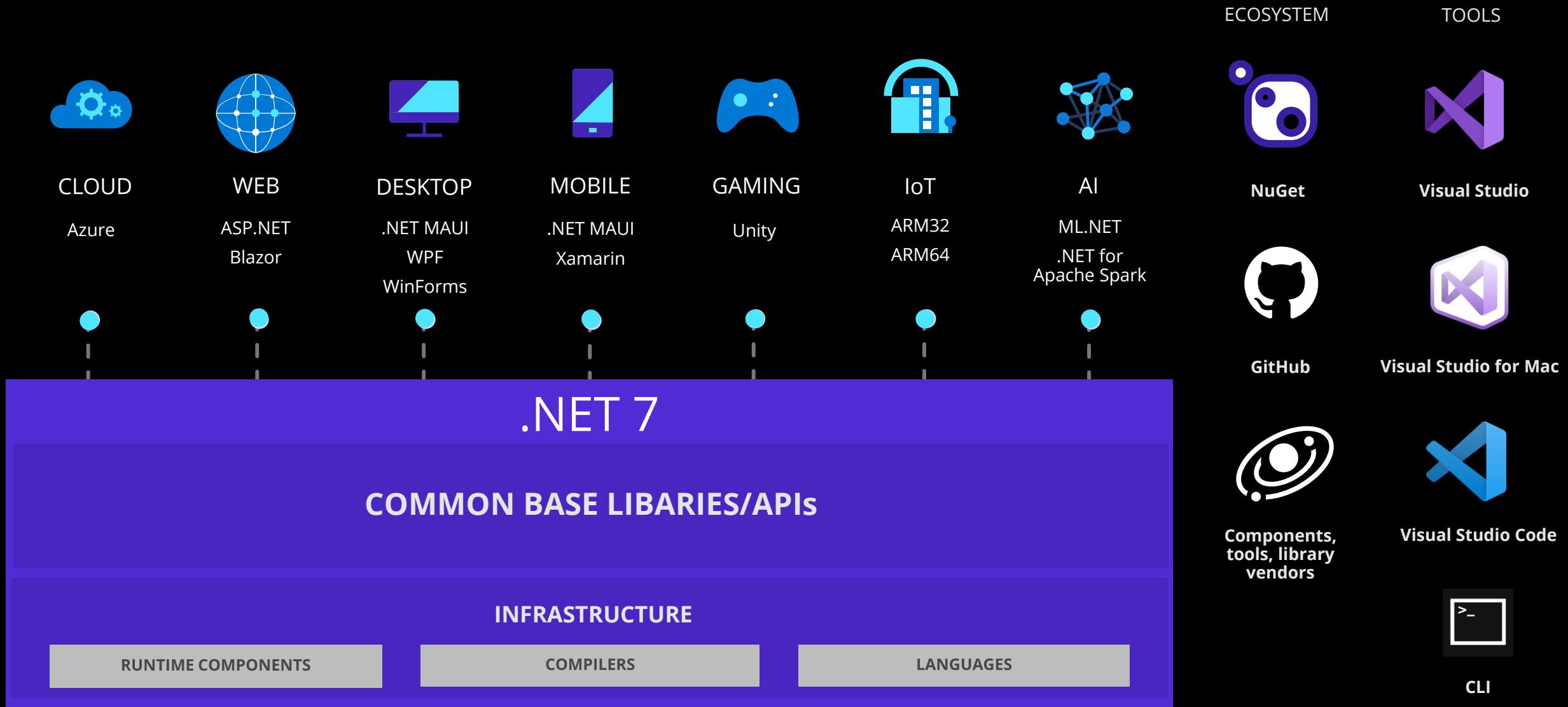
What is Session 1 about?

- Learning about .NET and running .NET locally and remotely on a Raspberry Pi
- Automating debugging with VS Code
- Getting started with Raspberry Pi and basic electronics
- Connecting your Raspberry Pi to Azure IoT and IoT Central

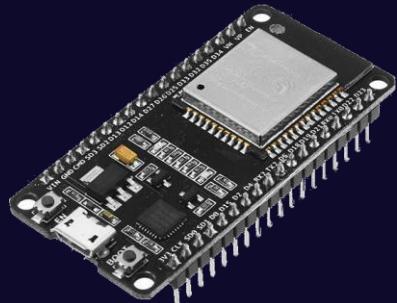


Foundation for more complex sessions

.NET - A unified development platform



Let's talk about devices



Devices

Microcomputers

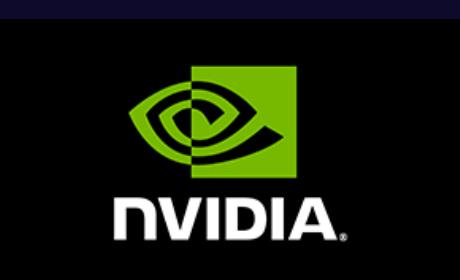
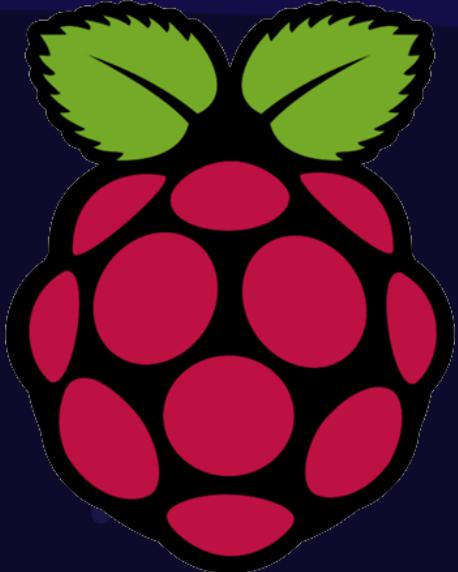
- Single board computers (SBC)
- X86 / 64 or ARM architecture
- RAM 512 MB+
- Operating System
 - Linux
 - Windows IoT
- High Power requirements
 - AI
 - Compute
- Interface with Hardware

Microcontrollers

- Small Single Chip Computer
- Dedicated Processor
- +- 64 K RAM
- No Operating System
- Low Power requirements
 - Long battery life
 - Remote locations
- Extremely Low Cost
 - High volume
- Interface with Hardware

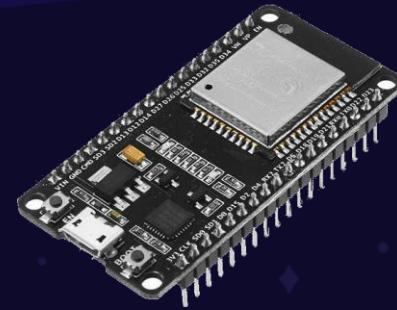
Microcomputers

- “Low-Cost” Computer
- Linux, Android, Windows 10
- IoT
- Interact with hardware and electronics
 - GPIO, I2C, SPI
 - HATs
- IoT Device (Internet of Things!)



Microcontrollers

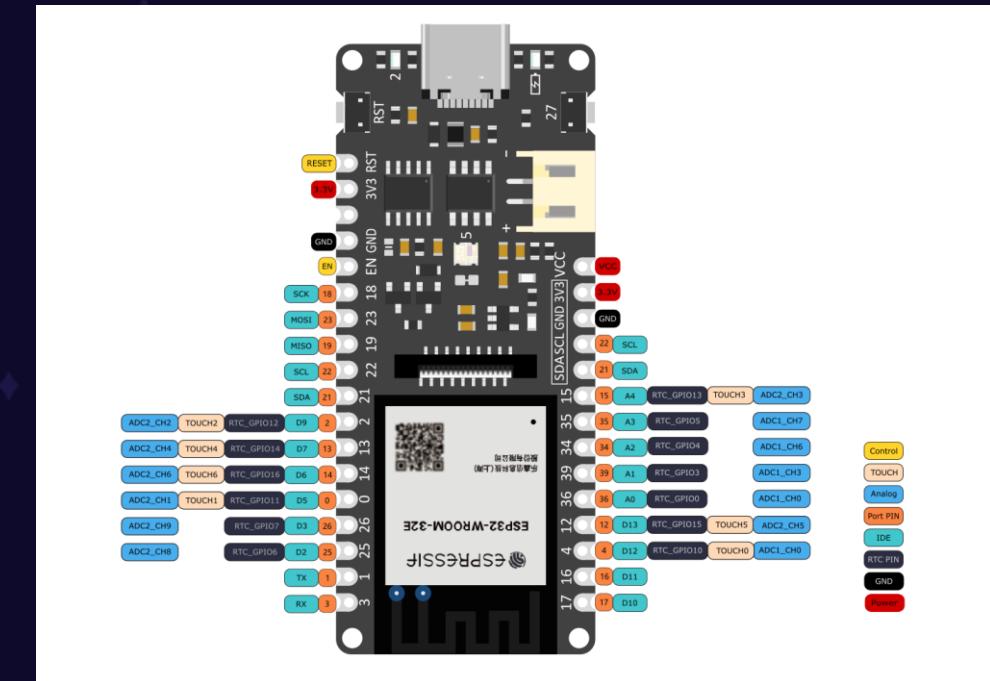
- Extremely low cost
- Long battery life
- Solutions run on the metal
- Interact with hardware and electronics
 - GPIO, I²C, SPI
 - HATs
- IoT Device (Internet of Things!)



Hardware Interfacing



Alternate Function	
I2C1 SDA	3.3V PWR
I2C1 SCL	GPIO 2
GPIO 3	5V PWR
GPIO 4	GND
GND	UART0 TX
GPIO 17	UART0 RX
GPIO 27	GPIO 18
GPIO 22	GND
3.3V PWR	GPIO 23
SPI0 MOSI	GPIO 24
SPI0 MISO	GPIO 10
SPI0 SCLK	GPIO 9
GND	GPIO 25
Reserved	GPIO 8
GPIO 5	SPI0 CS0
GPIO 6	GPIO 7
GPIO 13	SPI0 CS1
SPI1 MISO	Reserved
GPIO 19	GPIO 30
GPIO 26	GND
GND	GPIO 12
SPI1 MOSI	GPIO 32
GPIO 21	GPIO 34
GND	SPI1 CS0
	SPI1 MOSI
	SPI1 SCLK



.NET for IoT

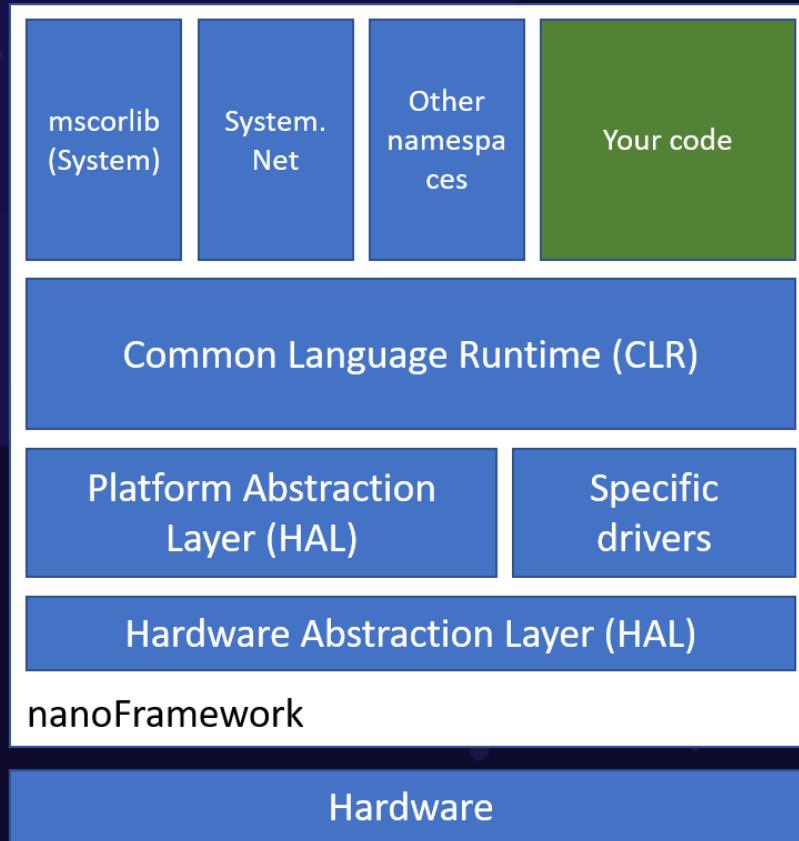


.NET 6 for ARM (v7+) or ARM64

- System.Device.Gpio
- IoT.Device.Bindings
- GPIO, UART, SPI, I2C

<https://dotnet.microsoft.com/apps/iot>

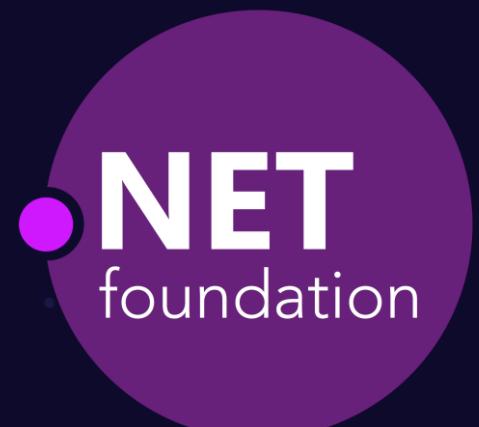
.NET Nanoframework



Reduced version of the .NET Common Language Runtime (CLR)
Subset of the .NET base class libraries
Resource-constrained devices with as low as 256kB of flash and 64kB of RAM
Runs directly on bare metal

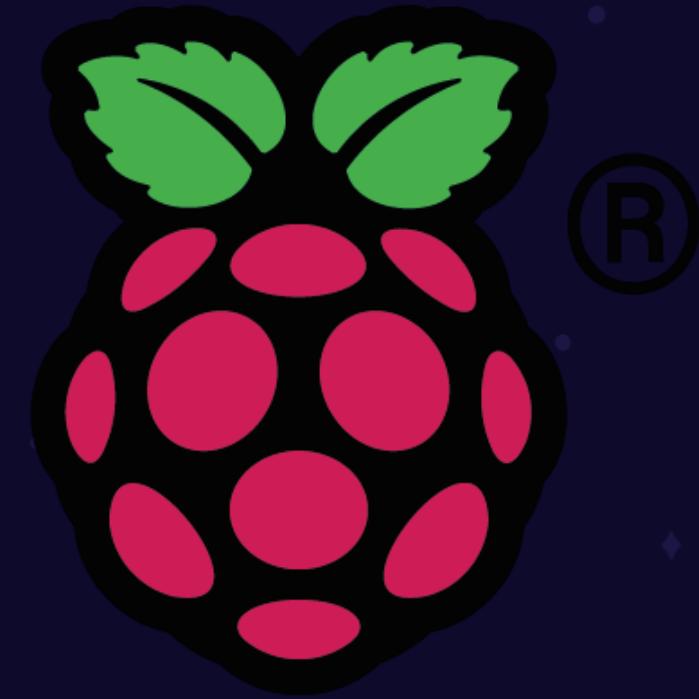
GPIO, UART, SPI, I2C

Espressif ESP32
OrgPal
STMicroelectronics
TI SimpleLink
NXP boards
Netduino boards



<https://docs.nanoframework.net/>

Module 1



Running a .NET Application locally and remotely on a
Raspberry Pi

Installing .NET 7

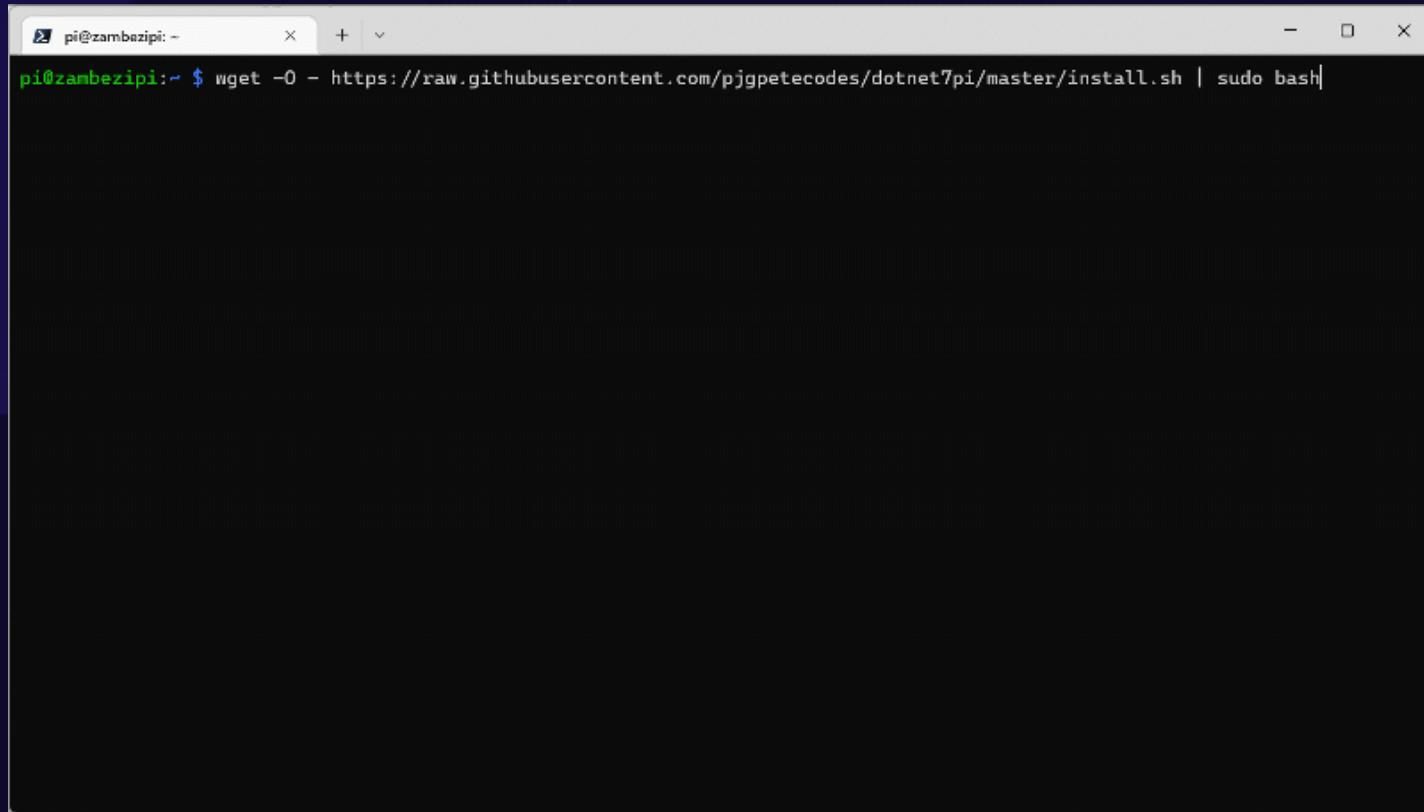
Build apps - SDK ⓘ

SDK 7.0.100

OS	Installers	Binaries
Linux	Package manager instructions	Arm32 Arm32 Alpine Arm64 Arm64 Alpine x64 x64 Alpine
macOS	Arm64 x64	Arm64 x64
Windows	Arm64 x64 x86 winget instructions	Arm64 x64 x86
All	dotnet-install scripts	

<https://dotnet.microsoft.com/en-us/download/dotnet/7.0>

Installing .NET 7 on Raspberry Pi



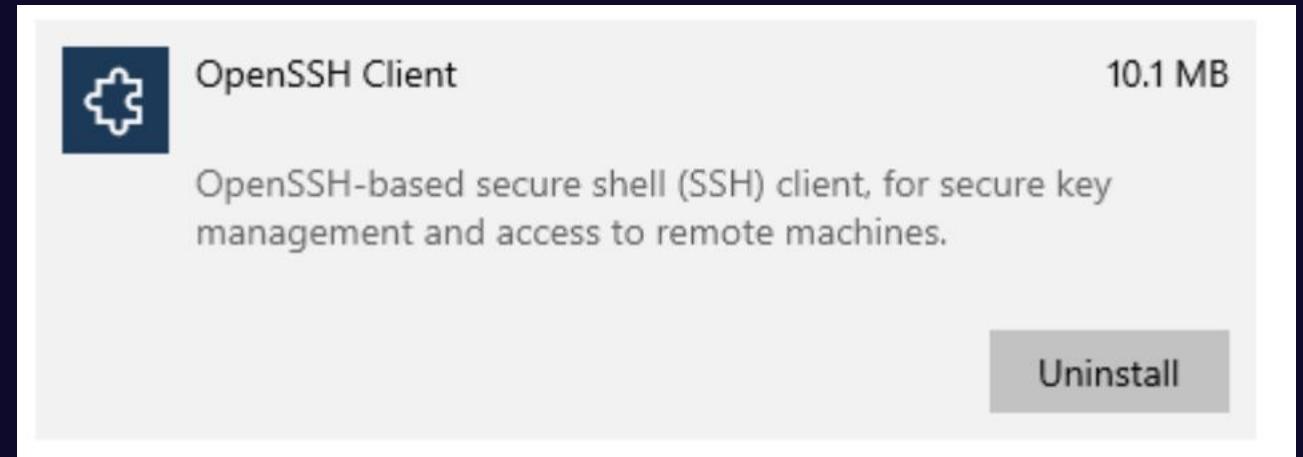
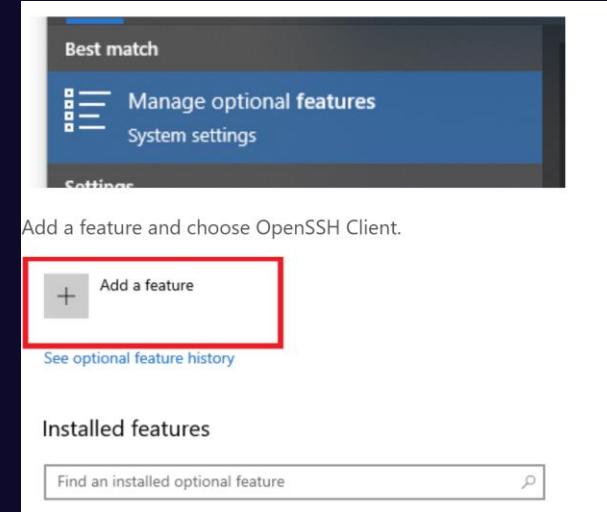
A screenshot of a terminal window titled "pi@zambezipi:~". The window contains a single line of text: "pi@zambezipi:~ \$ wget -O - https://raw.githubusercontent.com/pjgpeticodes/dotnet7pi/master/install.sh | sudo bash". The terminal has a dark background with white text.

```
wget -O - https://raw.githubusercontent.com/pjgpeticodes/dotnet7pi/master/install.sh | sudo bash
```

Development Tools

SSH

SSH, also known as Secure Shell or Secure Socket Shell, is a network protocol that gives users, a secure way to access a computer over a network.



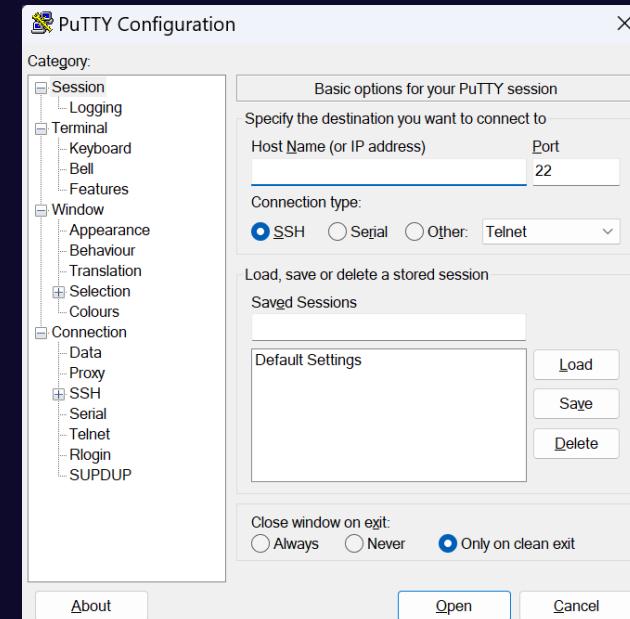
Development Tools

SSH

SSH, also known as Secure Shell or Secure Socket Shell, is a network protocol that gives users a secure way to access a computer over a network.

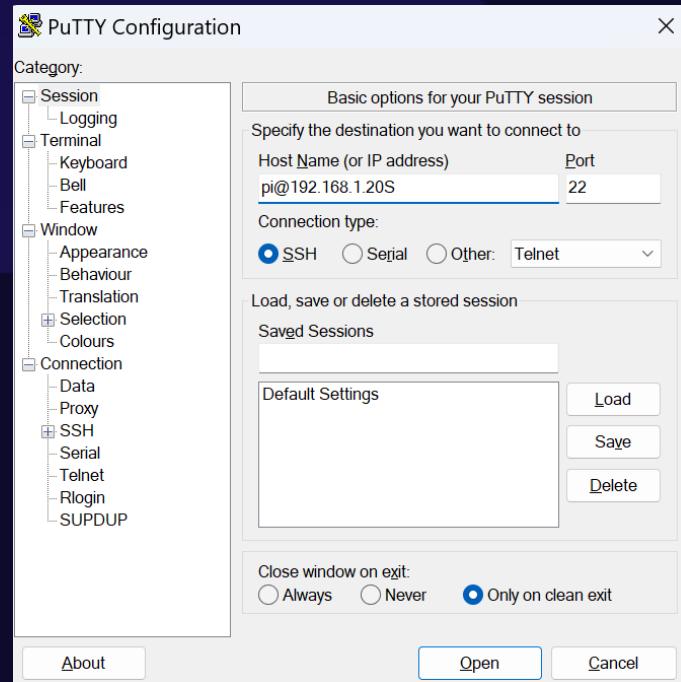
Putty

[Download PuTTY - a free SSH and telnet client for Windows](#)



Development Tools

SSH



Try it out!!

Switch your Raspberry PI on

Run Putty

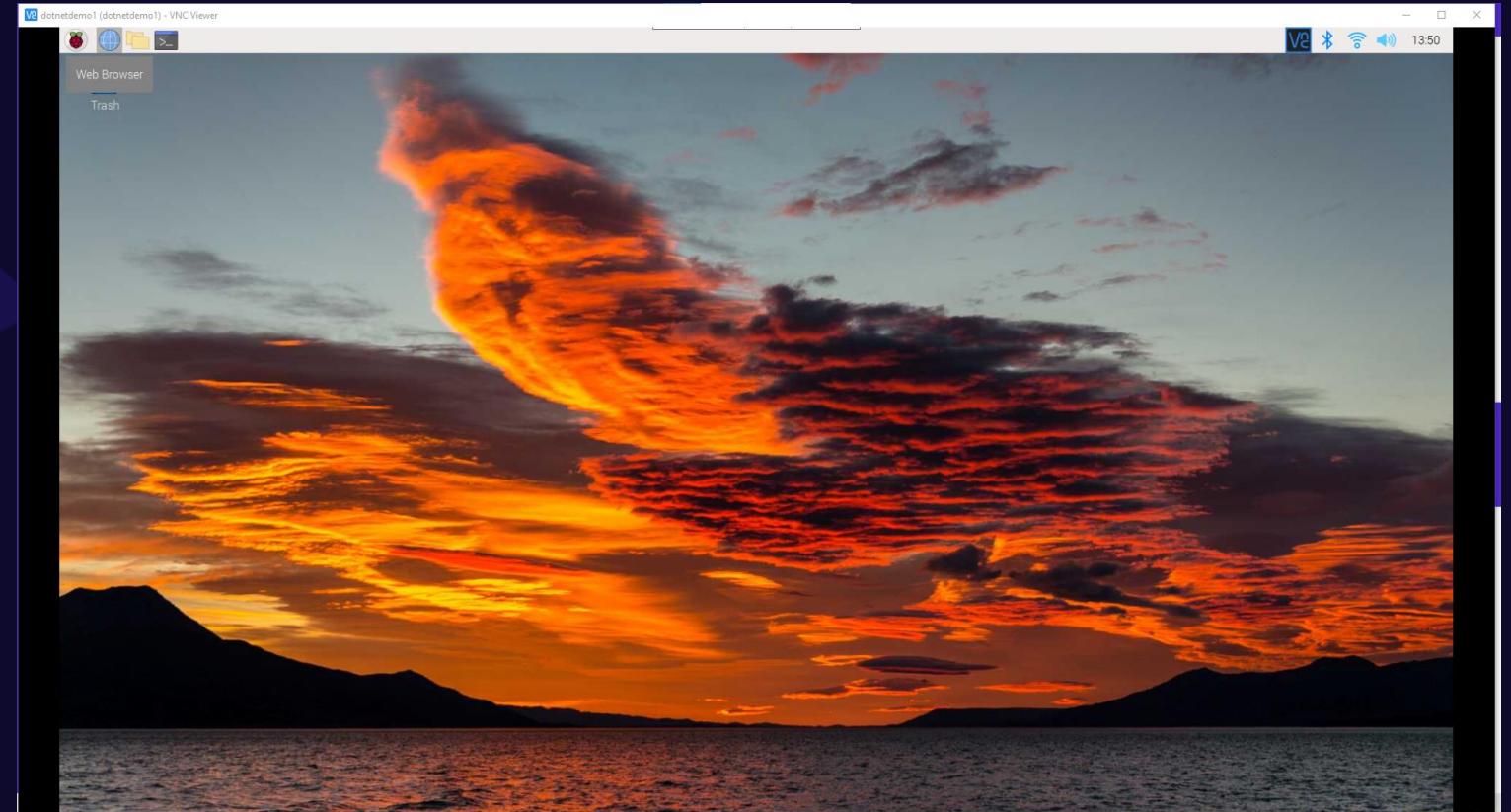
Connect to your PI with: pi@ipaddress

Password: mxchip12345

Development Tools

VNC

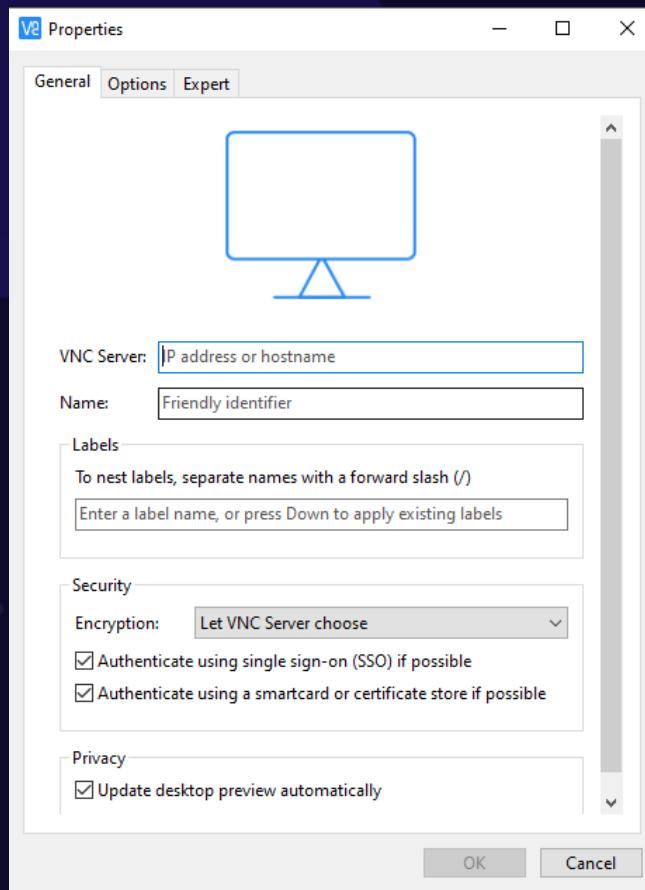
Virtual network computing (VNC) is a type of remote-control software that makes it possible to control another computer over a network connection.



<https://www.realvnc.com/en/connect/download/viewer/>

Development Tools

VNC



Try it out!!

Switch your Raspberry PI on

Run VNC

Connect to your PI with the IP Address

User Name: pi

Password: mxchip12345

<https://www.realvnc.com/en/connect/download/viewer/>

Development Tools

WinSCP

WinSCP is a popular SFTP client and FTP client for Microsoft Windows! Copy file between a local computer and remote servers using FTP, FTPS, SCP, SFTP, WebDAV or S3 file transfer protocols.

/home/pi/					
Name	Size	Changed	Rights	Owner	
..		11/13/2022 9:25:33 AM	rwxr-xr-x	root	
Bookshelf		9/22/2022 2:14:33 AM	rwxr-xr-x	pi	
cptmsdugs1		11/13/2022 9:57:35 PM	rwxr-xr-x	pi	
Desktop		11/13/2022 9:25:31 AM	rwxr-xr-x	pi	
Documents		11/13/2022 9:25:33 AM	rwxr-xr-x	pi	
dotnettest		11/13/2022 9:51:36 PM	rwxr-xr-x	pi	
Downloads		11/13/2022 9:25:33 AM	rwxr-xr-x	pi	
Music		11/13/2022 9:25:33 AM	rwxr-xr-x	pi	
Pictures		11/13/2022 9:25:33 AM	rwxr-xr-x	pi	
Public		11/13/2022 9:25:33 AM	rwxr-xr-x	pi	
Templates		11/13/2022 9:25:33 AM	rwxr-xr-x	pi	
Videos		11/13/2022 9:25:33 AM	rwxr-xr-x	pi	
2022-11-15-113945_1920x1080_scro...	1,823 KB	11/15/2022 11:39:46 AM	rw-r--r--	pi	
dotnetdebug.sh	1 KB	11/13/2022 9:51:04 AM	rwxr-xr-x	root	

<https://winscp.net/eng/index.php>

Development Tools

WinSCP

Try it out!!

Switch your Raspberry PI on

Run WinSCP

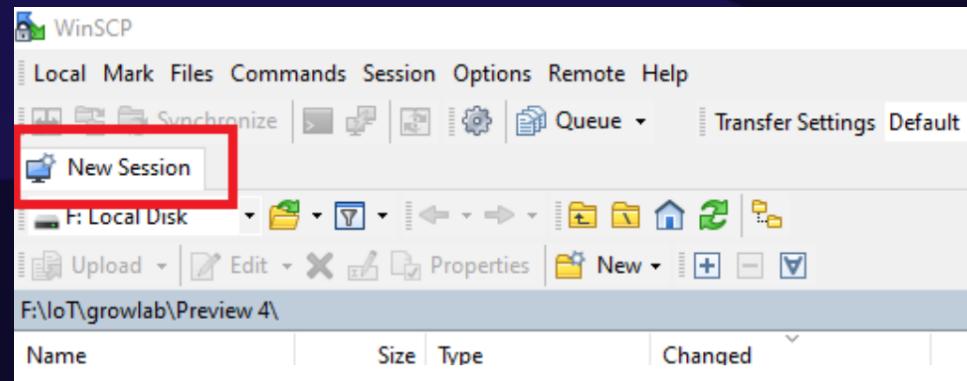
Connect to your PI with the IP Address

User Name: pi

Password: mxchip12345

Development Tools

WinSCP



Try it out!!

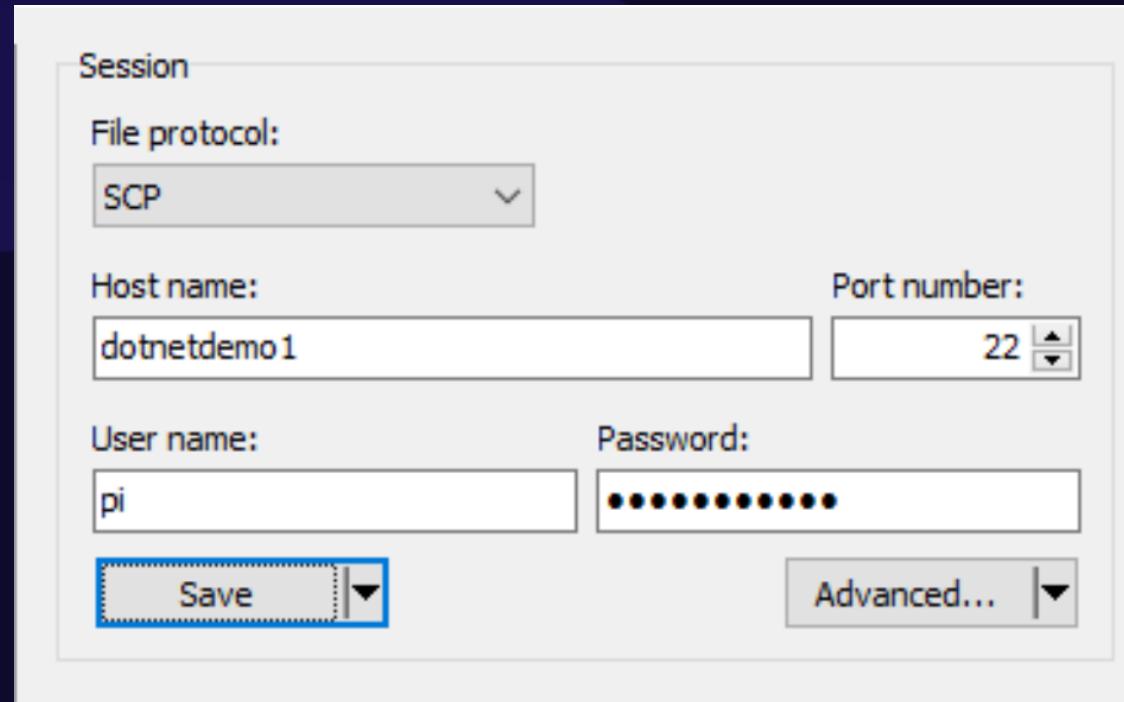
Switch your Raspberry PI on

Run WinSCP

Create New Session

Development Tools

WinSCP



Protocol SCP

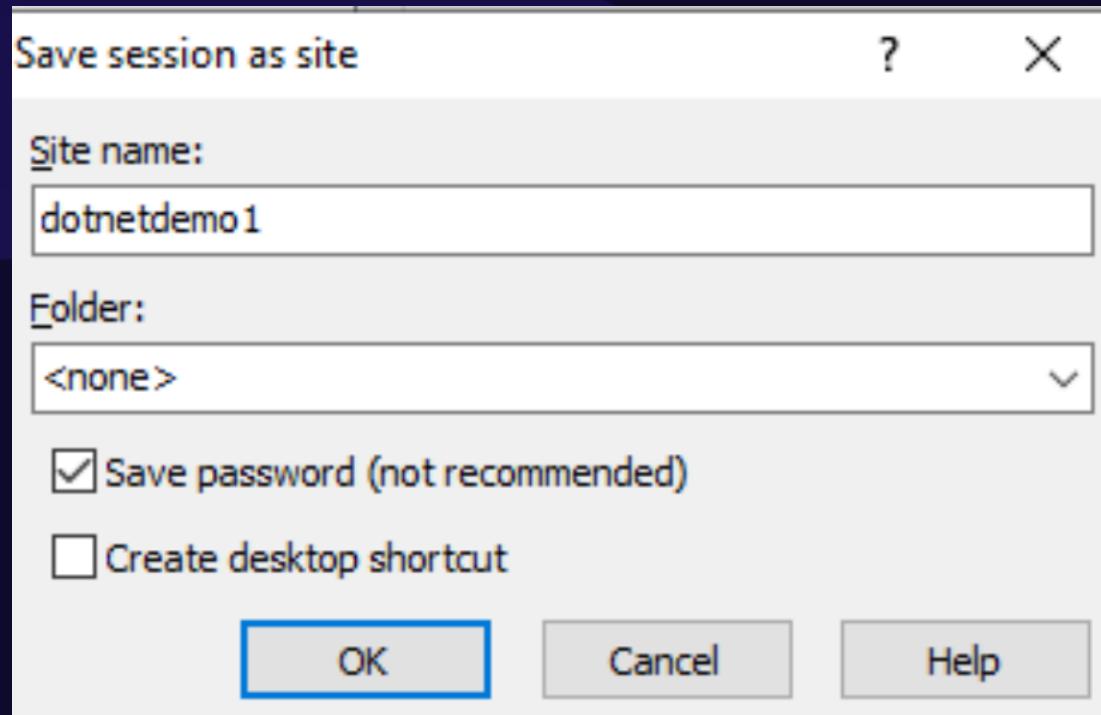
Host name: your ipaddress

Username: pi

Password: mxchip12345

Development Tools

WinSCP



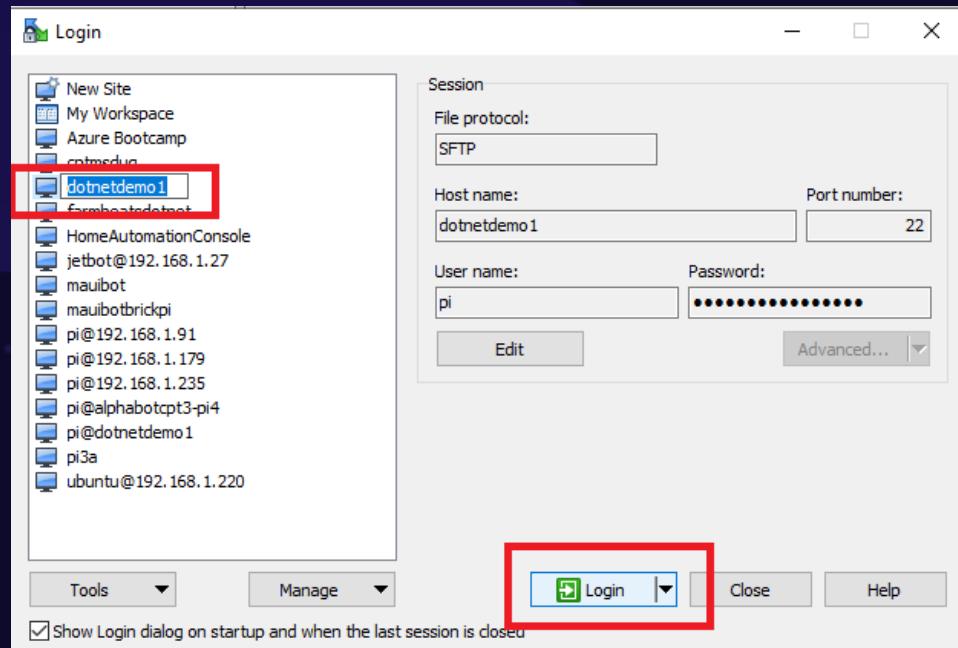
Save

Site name: The name of your pi. Use the sticker on the pi, eg: E1163323

Save Password

Development Tools

WinSCP



Login

Choose the name of your pi from the sites.

Click login

Development Tools

WinSCP

/home/pi/					
Name	Size	Changed	Rights	Owner	
..		11/13/2022 9:25:33 AM	rwxr-xr-x	root	
Bookshelf		9/22/2022 2:14:33 AM	rwxr-xr-x	pi	
cptmsdugs1		11/13/2022 9:57:35 PM	rwxr-xr-x	pi	
Desktop		11/13/2022 9:25:31 AM	rwxr-xr-x	pi	
Documents		11/13/2022 9:25:33 AM	rwxr-xr-x	pi	
dotnettest		11/13/2022 9:51:36 PM	rwxr-xr-x	pi	
Downloads		11/13/2022 9:25:33 AM	rwxr-xr-x	pi	
Music		11/13/2022 9:25:33 AM	rwxr-xr-x	pi	
Pictures		11/13/2022 9:25:33 AM	rwxr-xr-x	pi	
Public		11/13/2022 9:25:33 AM	rwxr-xr-x	pi	
Templates		11/13/2022 9:25:33 AM	rwxr-xr-x	pi	
Videos		11/13/2022 9:25:33 AM	rwxr-xr-x	pi	
2022-11-15-113945_1920x1080_scro...	1,823 KB	11/15/2022 11:39:46 AM	rw-r--r--	pi	
dotnetdebug.sh	1 KB	11/13/2022 9:51:04 AM	rwxr-xr-x	root	

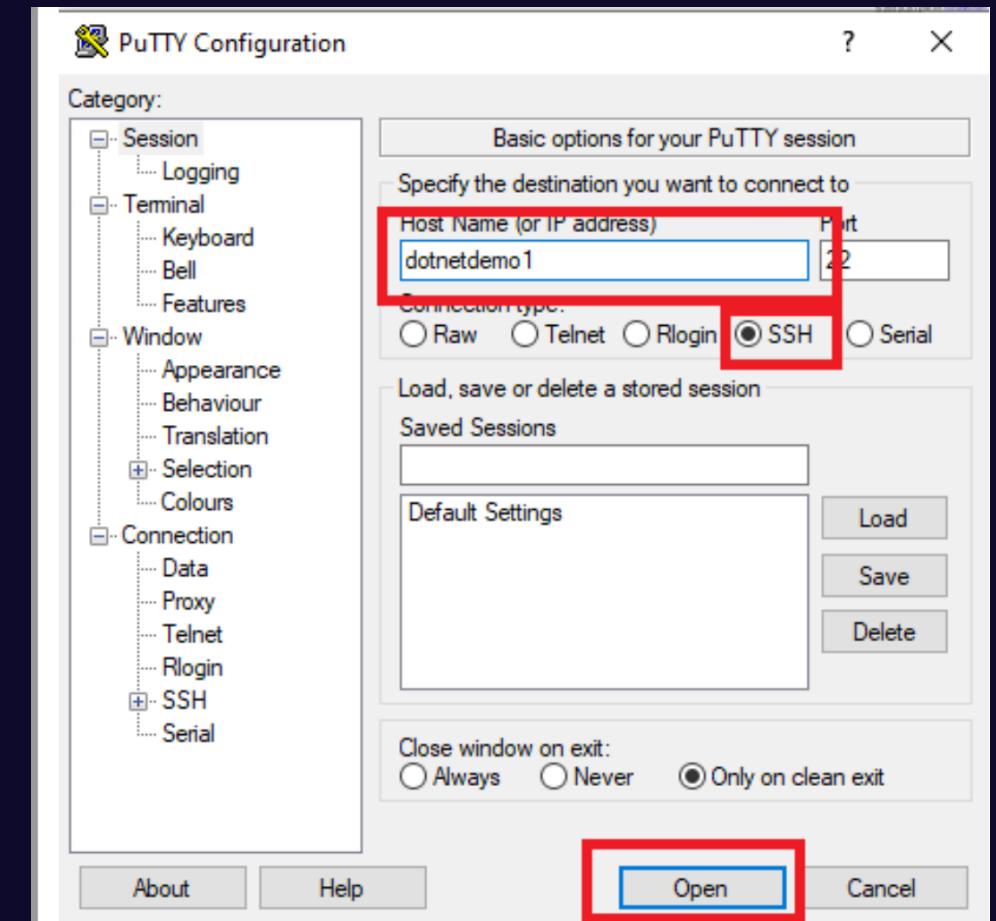
Running your First .NET 7 Application

Run Putty

Host Name: your pi's IP Address

Choose SSH

Open



Running your First .NET 7 Application

User Name: pi

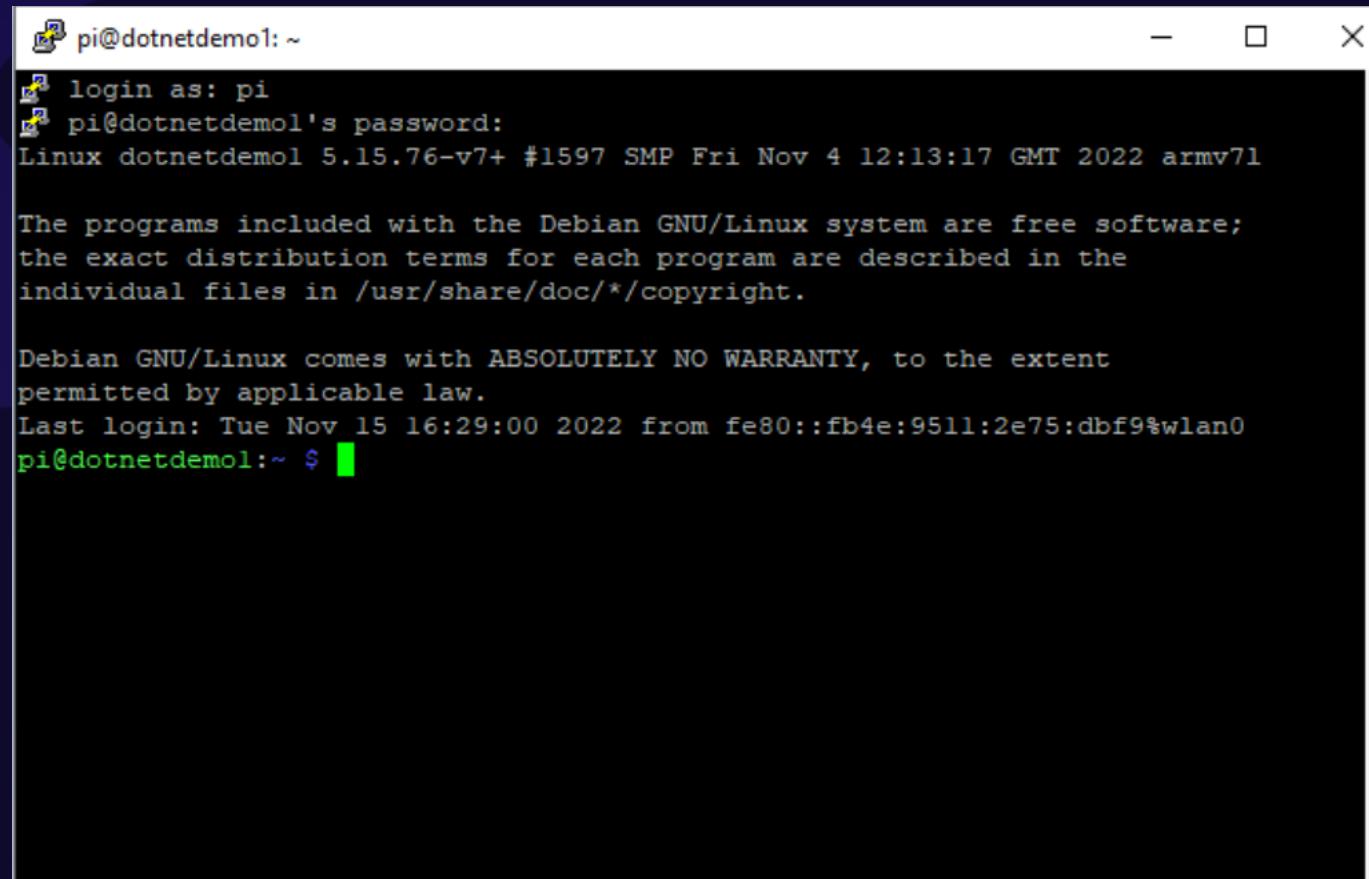
Password:
mxchip12345



A screenshot of a PuTTY terminal window titled "dotnetdemo1.local - PuTTY". The window shows the command "login as: pi" followed by "pi@dotnetdemol's password:" and a redacted password field.

```
dotnetdemo1.local - PuTTY
login as: pi
pi@dotnetdemol's password: [REDACTED]
```

Running your First .NET 7 Application



A screenshot of a terminal window titled "pi@dotnetdemol: ~". The window shows a Linux login process:

```
pi@dotnetdemol: ~
pi@dotnetdemol: ~$ login as: pi
pi@dotnetdemol's password:
Linux dotnetdemol 5.15.76-v7+ #1597 SMP Fri Nov 4 12:13:17 GMT 2022 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Nov 15 16:29:00 2022 from fe80::fb4e:9511:2e75:dbf9%wlan0
pi@dotnetdemol: ~ $
```

Running your First .NET 7 Application

Create a new directory
/ folder on the Pi

mkdir myfirstapp

```
pi@dotnetdemol:~ $ mkdir myfirstapp
pi@dotnetdemol:~ $ cd myfirstapp
pi@dotnetdemol:~/myfirstapp $
```

Running your First .NET 7 Application

Create a new directory
/ folder on the Pi

mkdir myfirstapp

Change to new
directory

cd myfirstapp

```
pi@dotnetdemol:~ $ mkdir myfirstapp
pi@dotnetdemol:~ $ cd myfirstapp
pi@dotnetdemol:~/myfirstapp $
```

Running your First .NET 7 Application

Create a new .NET 7
console app.

dotnet new console

```
pi@dotnetdemol:~/myfirstapp $ dotnet new console
The template "Console App" was created successfully.

Processing post-creation actions...
Restoring /home/pi/myfirstapp/myfirstapp.csproj:
  Determining projects to restore...
    Restored /home/pi/myfirstapp/myfirstapp.csproj (in 660 ms).
Restore succeeded.
```

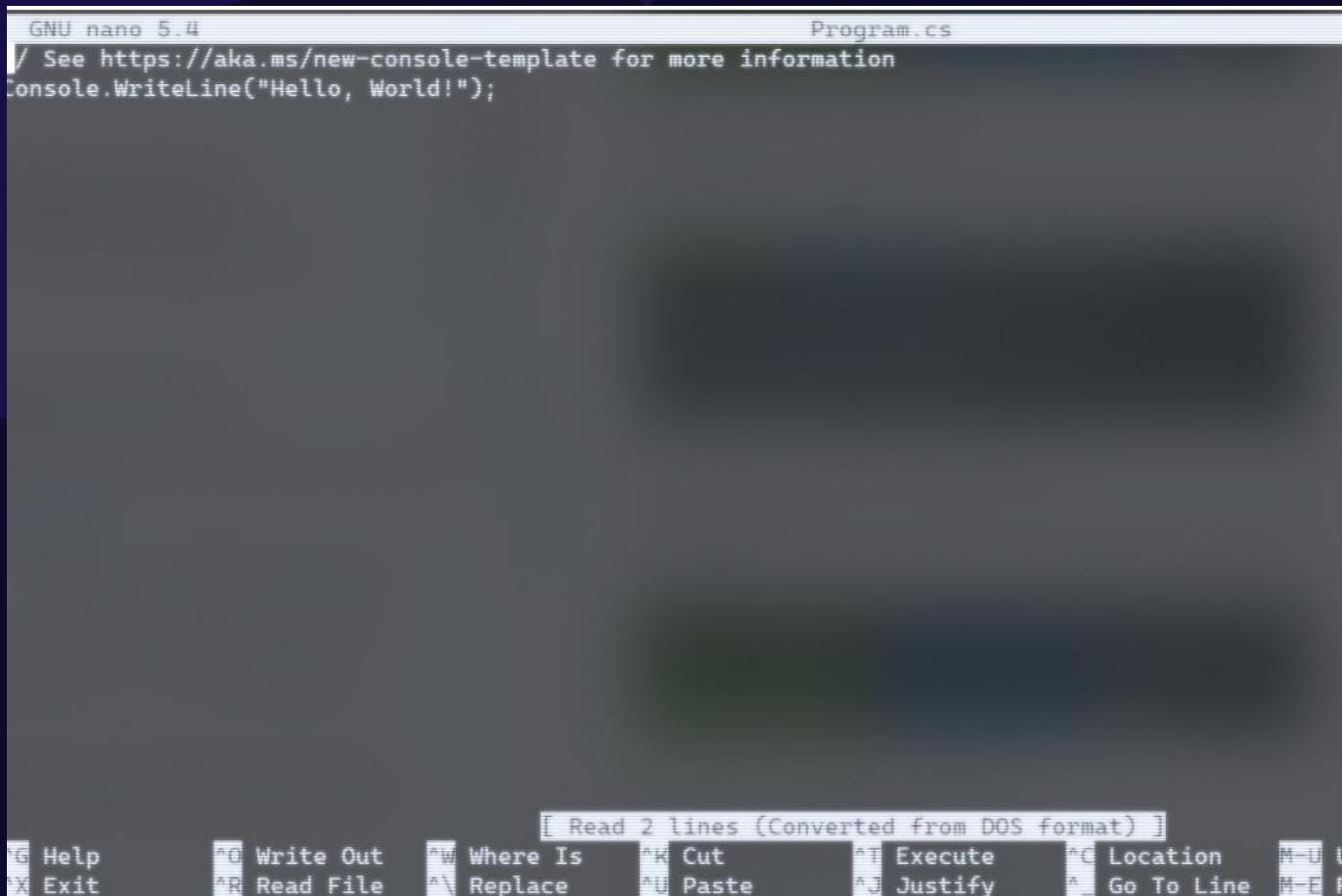
Running your First .NET 7 Application

Run your first .NET 7
app on a Raspberry Pi!

dotnet run

```
pi@dotnetdemol:~/myfirstapp $ dotnet run
Hello, World!
pi@dotnetdemol:~/myfirstapp $ █
```

Running your First .NET 7 Application



The screenshot shows a terminal window with the title "GNU nano 5.4" and the file name "Program.cs". The code in the editor is:

```
Program.cs
/ See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
```

At the bottom of the terminal window, there is a menu bar with the following options:

- G Help
- W Write Out
- W Where Is
- C Cut
- E Execute
- L Location
- M-U Undo
- X Exit
- R Read File
- R Replace
- P Paste
- J Justify
- G Go To Line
- M-E Redo

A status message at the bottom center of the terminal window reads "[Read 2 lines (Converted from DOS format)]".

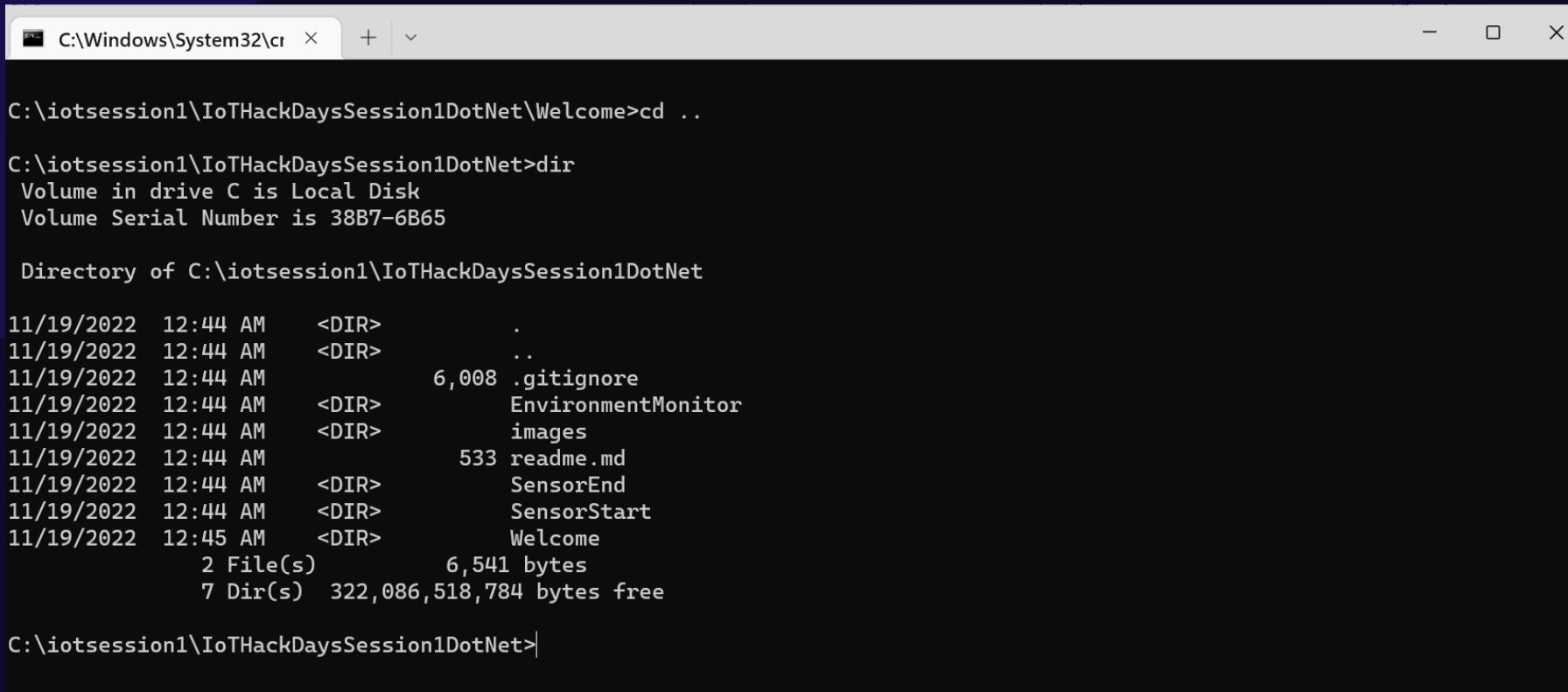
Running and creating your first .NET app on a REMOTE Raspberry pi

Remote development using a PC is much more efficient as the resources available (CPU, RAM, Disk space and Disk Speed) is so much more than an IoT device.

- Make a folder: c:\iotsession1
- Using command line: Clone the IoT Hack Session 1 Git Repository into the folder above
 - `git clone https://github.com/CPTMSDUG/IoTHackDaysSession1DotNet`
 -

<https://github.com/CPTMSDUG/IoTHackDaysSession1DotNet>

Running and creating your first .NET app on a REMOTE Raspberry pi



```
C:\Windows\System32\cr <--> + | x

C:\iotsession1\IoTHackDaysSession1DotNet\Welcome>cd ..

C:\iotsession1\IoTHackDaysSession1DotNet>dir
Volume in drive C is Local Disk
Volume Serial Number is 38B7-6B65

Directory of C:\iotsession1\IoTHackDaysSession1DotNet

11/19/2022  12:44 AM    <DIR>        .
11/19/2022  12:44 AM    <DIR>        ..
11/19/2022  12:44 AM            6,008 .gitignore
11/19/2022  12:44 AM    <DIR>        EnvironmentMonitor
11/19/2022  12:44 AM    <DIR>        images
11/19/2022  12:44 AM            533 readme.md
11/19/2022  12:44 AM    <DIR>        SensorEnd
11/19/2022  12:44 AM    <DIR>        SensorStart
11/19/2022  12:45 AM    <DIR>        Welcome
                           2 File(s)      6,541 bytes
                           7 Dir(s)   322,086,518,784 bytes free

C:\iotsession1\IoTHackDaysSession1DotNet>
```

<https://github.com/CPTMSDUG/IoTHackDaysSession1DotNet>

Running and creating your first .NET app on a REMOTE Raspberry pi

Welcome Project

Change to the Welcome folder

cd Welcome

```
C:\iotsession1\IoTHackDaysSession1DotNet>cd welcome

C:\iotsession1\IoTHackDaysSession1DotNet\Welcome>dir
 Volume in drive C is Local Disk
 Volume Serial Number is 38B7-6B65

Directory of C:\iotsession1\IoTHackDaysSession1DotNet\Welcome

11/19/2022  12:45 AM    <DIR>        .
11/19/2022  12:44 AM    <DIR>        ..
11/19/2022  12:44 AM            39  .env
11/19/2022  12:44 AM    <DIR>        .vscode
11/19/2022  12:45 AM    <DIR>        bin
11/19/2022  12:44 AM            47  global.json
11/19/2022  12:44 AM            444 netpublish.bat
11/19/2022  12:45 AM    <DIR>        obj
11/19/2022  12:44 AM            301 Program.cs
11/19/2022  12:44 AM            392 Welcome.csproj
11/19/2022  12:44 AM            522 welcometext.txt
                           6 File(s)        1,745 bytes
                           5 Dir(s)  322,083,233,792 bytes free
```

<https://github.com/CPTMSDUG/IoTHackDaysSession1DotNet>

Running and creating your first .NET app on a REMOTE Raspberry pi

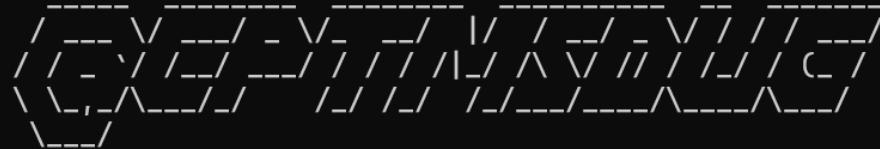
Welcome Project

Run the project

dotnet run

This is running on your laptop!

```
C:\iotsession1\IoTHackDaysSession1DotNet>Welcome>dotnet run
```



```
Welcome to IoT Hack Days Session 1!
```

```
:~~. :~~.  
. \ ' ' / .'  
.~ .~~~..~.  
: .~.'~'.~. :  
~( ) ( ) ~  
( : '~~'.~.'~~' : )  
~ .~ ( ) ~. ~  
( : '~~' : ) Raspberry Pi  
'~~ .~~~. ~'  
'~~'
```

```
This is a .NET app.
```

<https://github.com/CPTMSDUG/IoTHackDaysSession1DotNet>

Running and creating your first .NET app on a REMOTE Raspberry pi

Welcome Project

Cross compile for an ARM processor

Raspberry Pi isn't an Intel x86, it's an ARM architecture device

```
dotnet publish -r linux-arm
```

<https://github.com/CPTMSDUG/IoTHackDaysSession1DotNet>

Running and creating your first .NET app on a REMOTE Raspberry pi

Welcome Project

Cross compile for an ARM processor

Not self contained. Relies on installed runtime

`dotnet publish -r linux-arm --no-self-contained`

Self contained. Runtime is packaged and deployed with the application

`dotnet publish -r linux-arm --self-contained`

<https://github.com/CPTMSDUG/IoTHackDaysSession1DotNet>

Running and creating your first .NET app on a REMOTE Raspberry pi

Welcome Project

Cross compile for an ARM processor

Try it out!

`dotnet publish -r linux-arm --self-contained`

```
C:\iotsession1\IoTHackDaysSession1DotNet\Welcome>dotnet publish -r linux-arm --self-contained
MSBuild version 17.3.2+561848881 for .NET
Determining projects to restore...
Restored C:\iotsession1\IoTHackDaysSession1DotNet\Welcome\Welcome.csproj (in 185 ms).
Welcome -> C:\iotsession1\IoTHackDaysSession1DotNet\Welcome\bin\Debug\net6.0\linux-arm\Welcome.dll
Welcome -> C:\iotsession1\IoTHackDaysSession1DotNet\Welcome\bin\Debug\net6.0\linux-arm\publish\
```

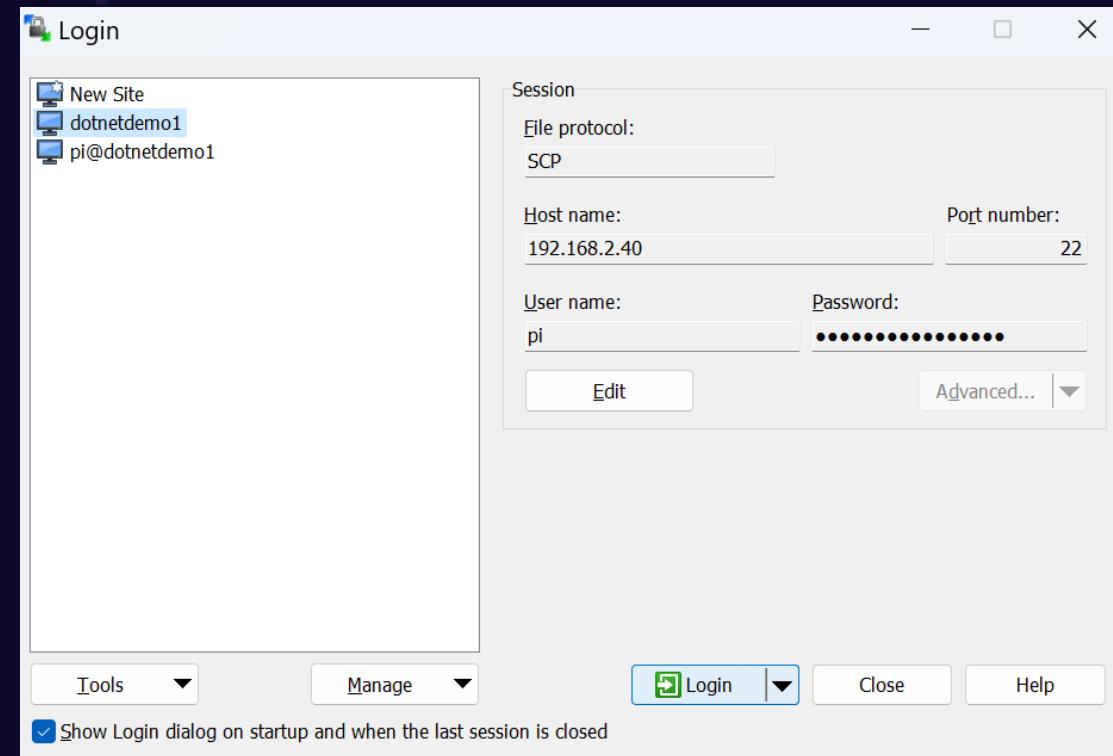
<https://github.com/CPTMSDUG/IoTHackDaysSession1DotNet>

Running and creating your first .NET app on a REMOTE Raspberry pi

Welcome Project

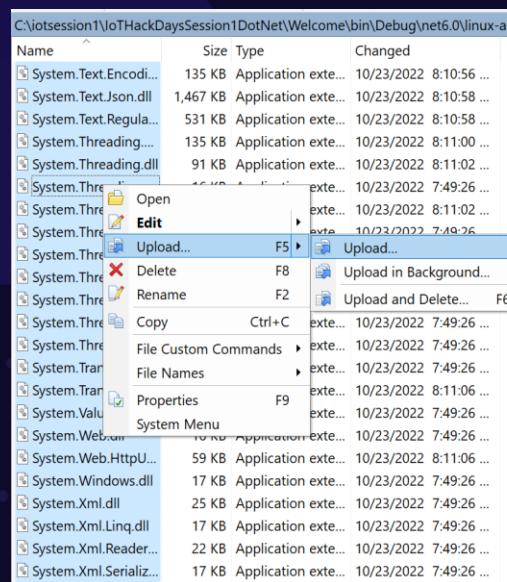
Cross compile for an ARM processor

Deploy to the Raspberry Pi with WinSCP

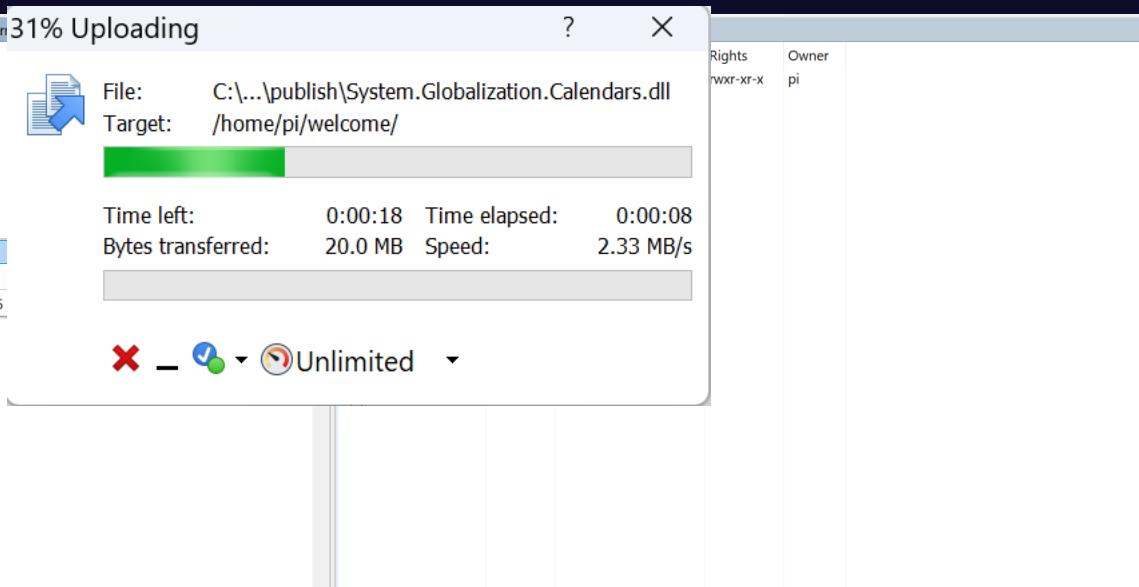


Running and creating your first .NET app on a REMOTE Raspberry pi

Local publish folder



Remote Pi folder



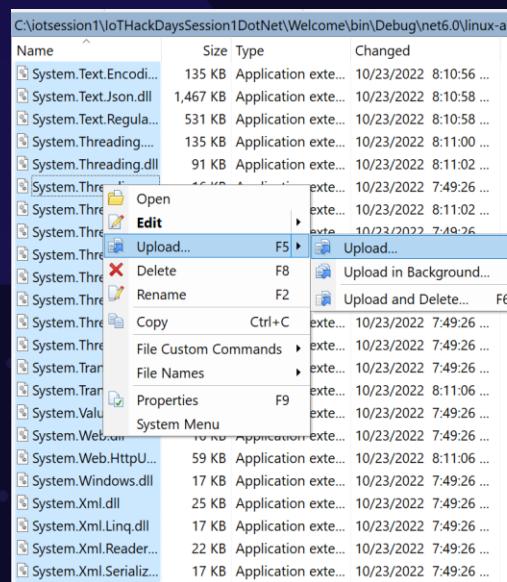
Select all files from the publish folder.

Select the Welcome folder on the right on the PI

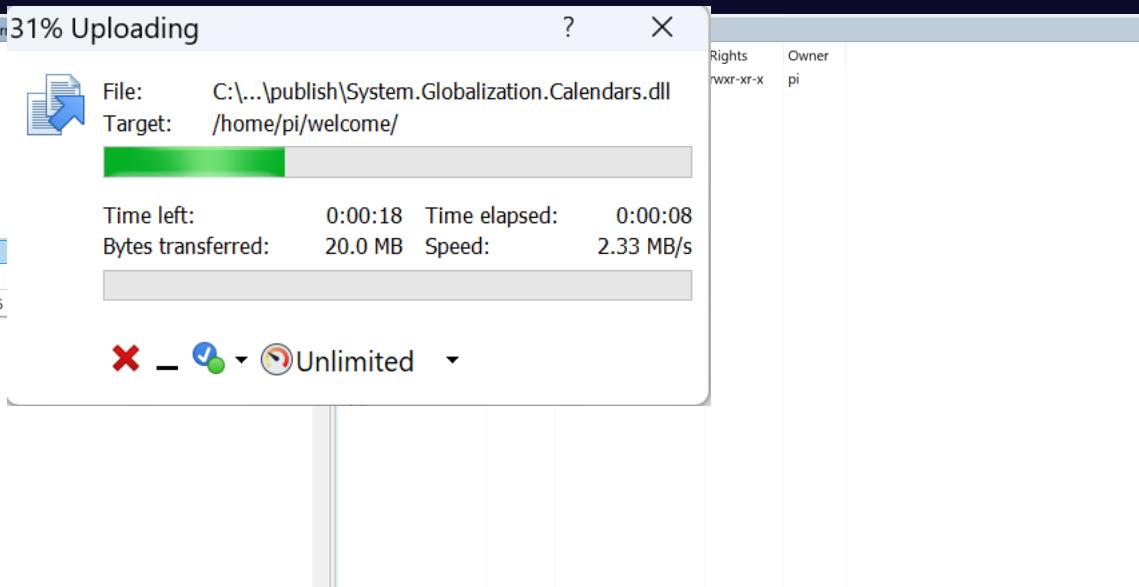
Right click: Upload

Running and creating your first .NET app on a REMOTE Raspberry pi

Local publish folder



Remote Pi folder



Select all files from the publish folder.

Select the Welcome folder on the right on the PI

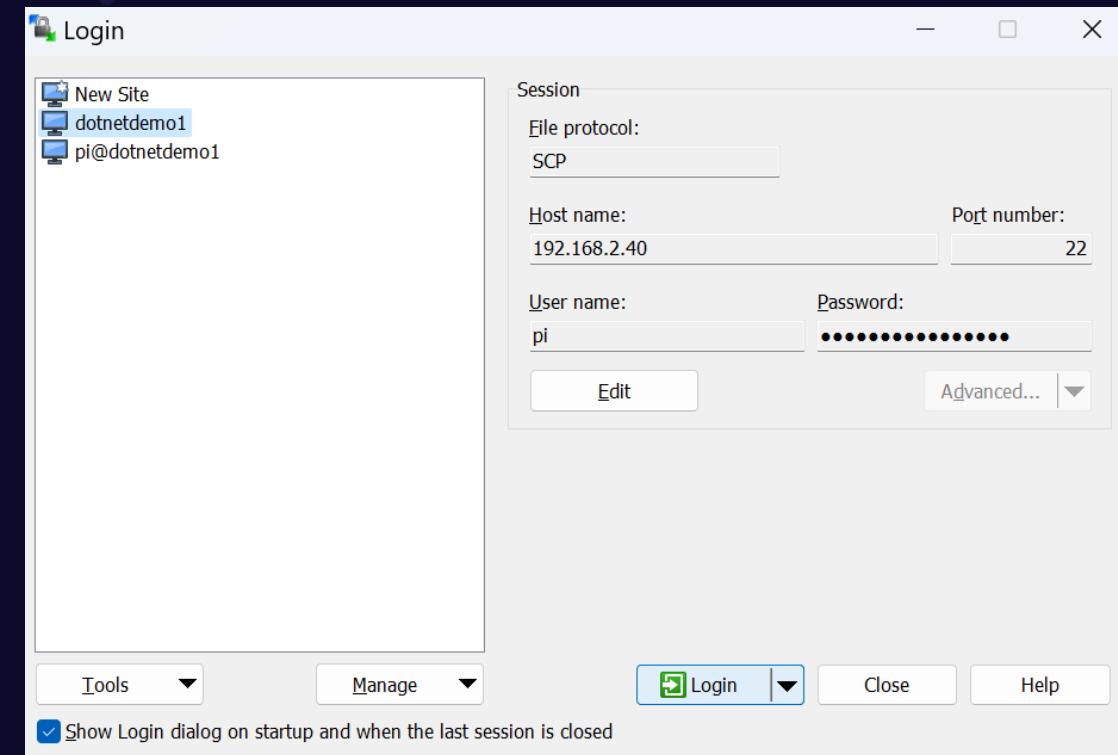
Right click: Upload

Running and creating your first .NET app on a REMOTE Raspberry pi

Welcome Project

Cross compile for an ARM processor

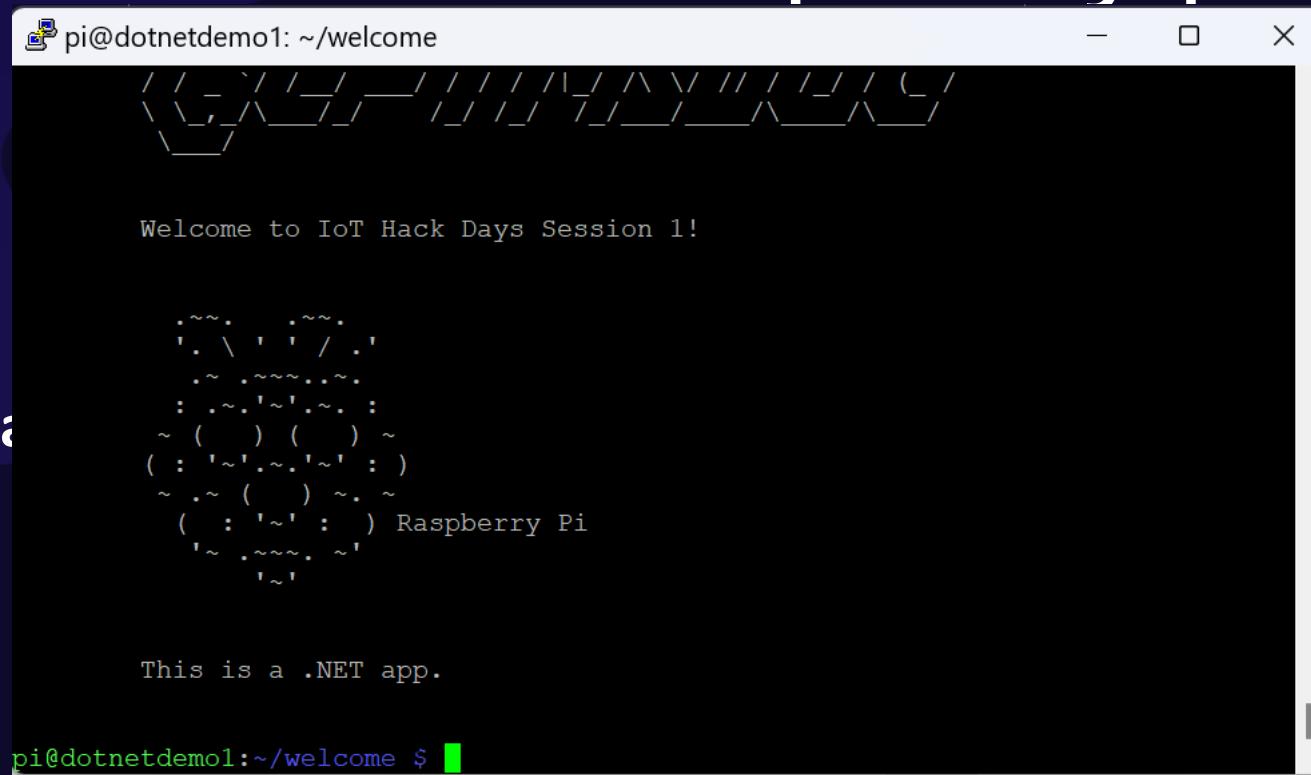
Run the application on the pi via SSH



Running and creating your first .NET app on a REMOTE Raspberry pi

Welcome Project

Cross compile for a processor



This is running on your Remote Raspberry Pi!

Visual Studio Code

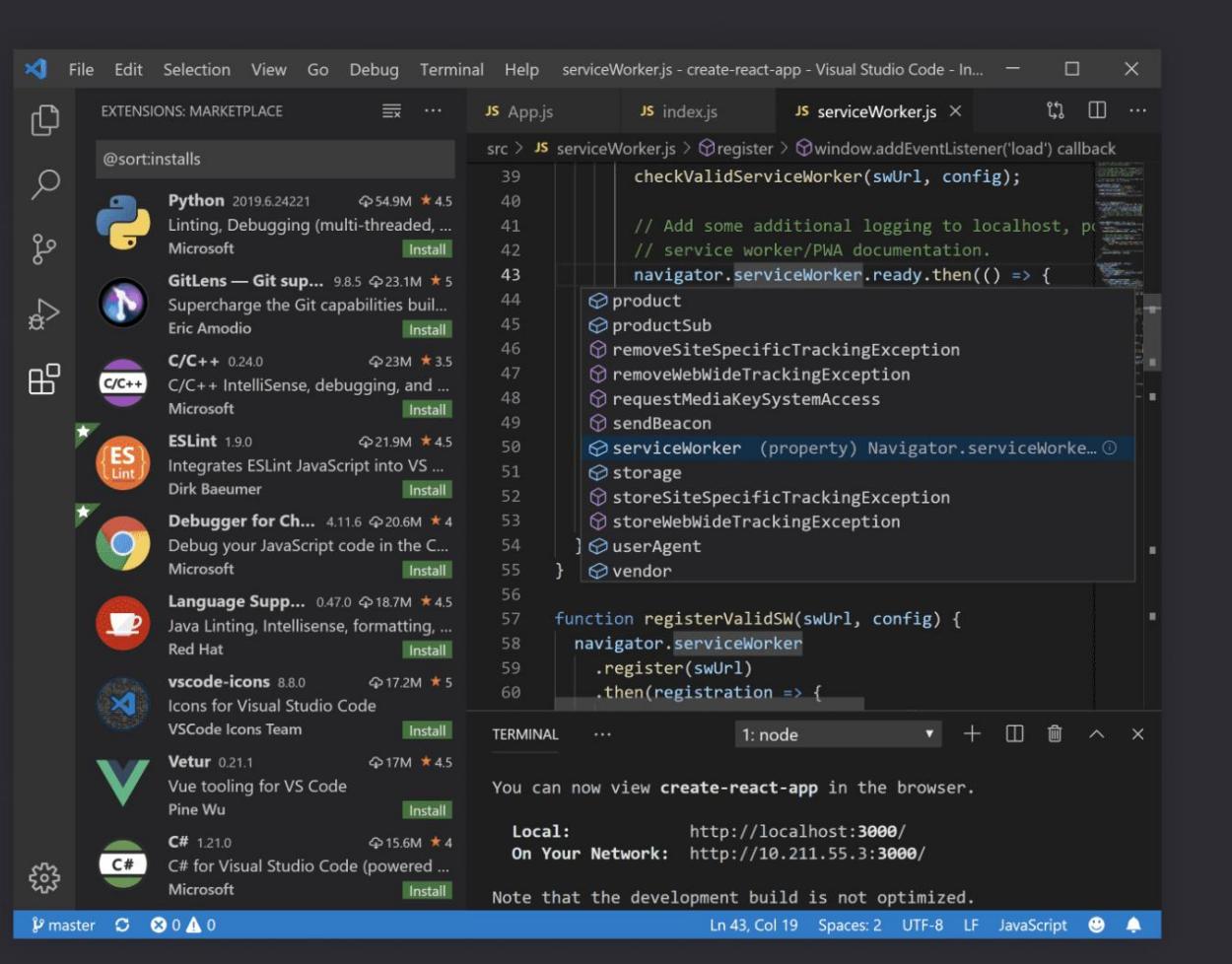
Code editing.
Redefined.

Free. Built on open source. Runs everywhere.

Download for Windows
Stable Build

Web, Insiders edition, or other platforms

By using VS Code, you agree to its
license and privacy statement.



<https://code.visualstudio.com/>

Visual Studio Code

Visual Studio Remote Debugger

Write code in
deploy, run and

```
pi@dotnetdemol:~ $ curl -sSL https://aka.ms/getvsdbgsh | /bin/sh /dev/stdin -v :  
atest -l ~/vsdbg  
Info: Previous installation at '/home/pi/vsdbg' not found  
Info: Using vsdbg version '17.4.11017.1'  
Info: Creating install directory  
Using arguments  
  Version          : 'latest'  
  Location         : '/home/pi/vsdbg'  
  SkipDownloads   : 'false'  
  LaunchVsDbgAfter: 'false'  
  RemoveExistingOnUpgrade : 'false'  
Info: Using Runtime ID 'linux-arm'  
Downloading https://vsdebugger.azureedge.net/vsdbg-17-4-11017-1/vsdbg-linux-arm.  
tar.gz  
Info: Successfully installed vsdbg at '/home/pi/vsdbg'
```

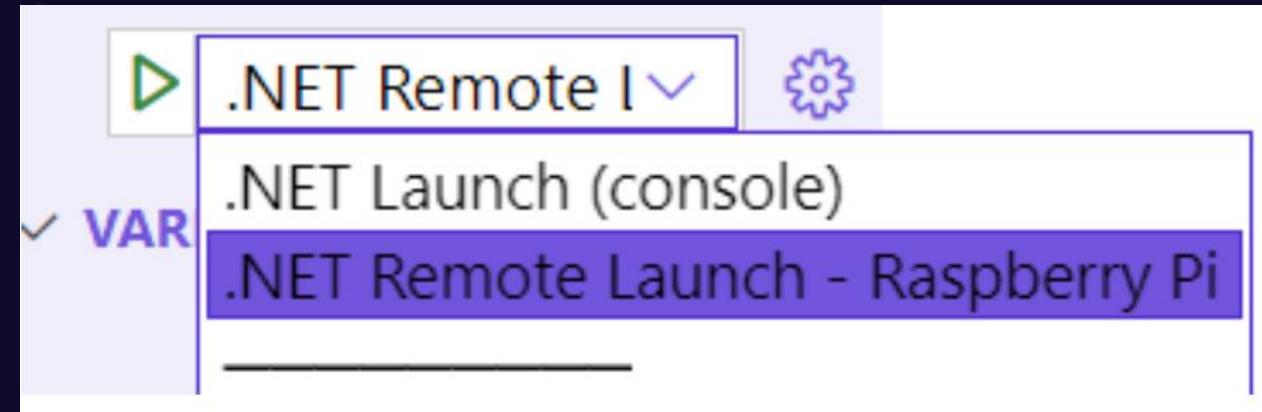
To install the Visual Studio Remote Debugger on your Raspberry pi, run the following command:

```
curl -sSL https://aka.ms/getvsdbgsh | /bin/sh /dev/stdin -v latest -l ~/vsdbg
```

Visual Studio Code

Automate Cross Compile and Deploy with Visual Studio code

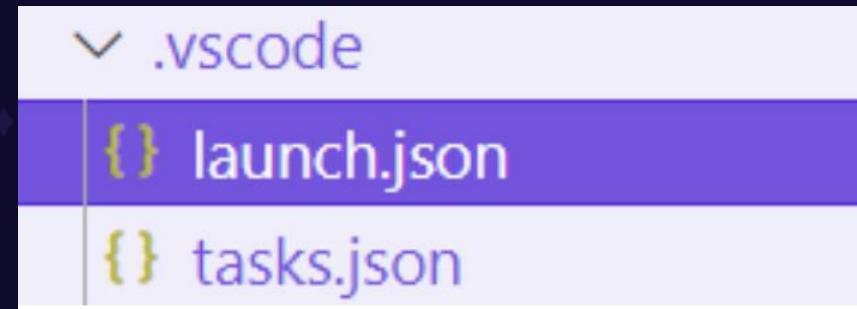
Launch Profiles



Visual Studio Code

Automate Cross Compile and Deploy with Visual Studio code

To create a launch configuration create a "launch.json" in the .vscode folder.



The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** Program.cs - Welcome - Visual Studio Code.
- Sidebar:** Explorer, Search, Problems, Databases, Task, Terminal, and Recent.
- Left Panel:** Shows a tree view of the project structure under "WELCOME". Items listed include .vscode, bin, obj, .env, global.json, netpublish.bat, Program.cs (which is selected), Welcome.csproj, and welcometext.txt.
- Code Editor:** Displays the content of Program.cs. The code reads lines from a file named "welcomete" and prints them to the console.

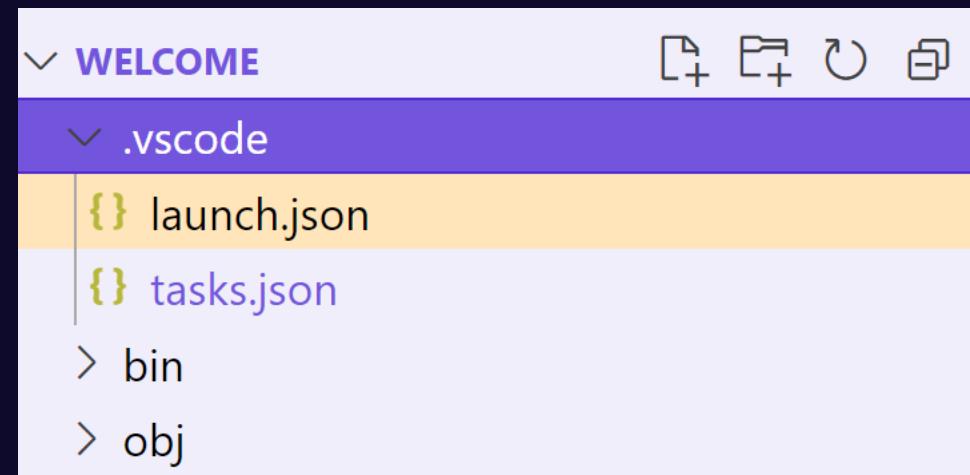
```
C# Program.cs X
C# Program.cs > {} Welcome
1  namespace Welcome;
2  class Program
3  {
4      static void Main(string[] args)
5      {
6          string[] lines = System.IO.File.ReadAllLines(@"welcomete");
7
8          foreach (string line in lines)
9          {
10              Console.WriteLine("\t" + line);
11          }
12
13
14
15      }
16  }
17
```

- Bottom Bar:** Open Recent and a right arrow icon.

Visual Studio Code

Automate Cross Compile and Deploy with Visual Studio code

Open the launch.json



Visual Studio Code

Automate Cross Compile and Deploy with Visual Studio code

Launch.json allows the setting up in one or many launch configurations.

It allows the execution of programs and also different tasks to be executed.

```
{
  "name": ".NET Remote Launch - Raspberry Pi",
  "type": "coreclr",
  "request": "launch",
  "preLaunchTask": "publish",
  "program": "dotnet",
  "args": ["./Welcome.dll"],
  "cwd": "/home/pi/welcome",
  "stopAtEntry": false,
  "console": "internalConsole",
  "pipeTransport": {
    "pipeCwd": "${workspaceRoot}",
    "pipeProgram": "c:\\\\Program Files\\\\PuTTY\\\\plink.exe",
    "pipeArgs": [
      "-pw",
      "${input:PiPassword}",
      "${input:PiHostName}"
    ],
    "debuggerPath": "sudo ~/vsdbg/vsdbg",
  }
}
```

Visual Studio Code

Automate Cross Compile and Deploy with Visual Studio code

Execute the dotnet command

Args parameter to run Welcome.dll

Cwd is the location on the pi

Pipetransport executes via SSH

using **Putty** and **plink**

DebuggerPath is path to VS Remote Debugger

```
{
  "name": ".NET Remote Launch - Raspberry Pi",
  "type": "coreclr",
  "request": "launch",
  "preLaunchTask": "publish",
  "program": "dotnet",
  "args": ["./Welcome.dll"],
  "cwd": "/home/pi/welcome",
  "stopAtEntry": false,
  "console": "internalConsole",
  "pipeTransport": {
    "pipeCwd": "${workspaceRoot}",
    "pipeProgram": "c:\\\\Program Files\\\\PuTTY\\\\plink.exe",
    "pipeArgs": [
      "-pw",
      "${input:PiPassword}",
      "${input:PiHostName}"
    ],
    "debuggerPath": "sudo ~/vsdbg/vsdbg"
  }
}
```

Visual Studio Code

Automate Cross Compile and Deploy with Visual Studio code

To complete the run experience
the app would need to be compile.

preLaunchTasks enable this.

```
{
  "name": ".NET Remote Launch - Raspberry Pi",
  "type": "coreclr",
  "request": "launch",
  "preLaunchTask": "publish",
  "program": "dotnet",
  "args": ["./Welcome.dll"],
  "cwd": "/home/pi/welcome",
  "stopAtEntry": false,
  "console": "internalConsole",
  "pipeTransport": {
    "pipeCwd": "${workspaceRoot}",
    "pipeProgram": "c:\\\\Program Files\\\\PuTTY\\\\plink.exe",
    "pipeArgs": [
      "-pw",
      "${input:PiPassword}",
      "${input:PiHostName}"
    ],
    "debuggerPath": "sudo ~/vsdbg/vsdbg",
  }
}
```

Visual Studio Code

Automate Cross Compile and Deploy with Visual Studio code

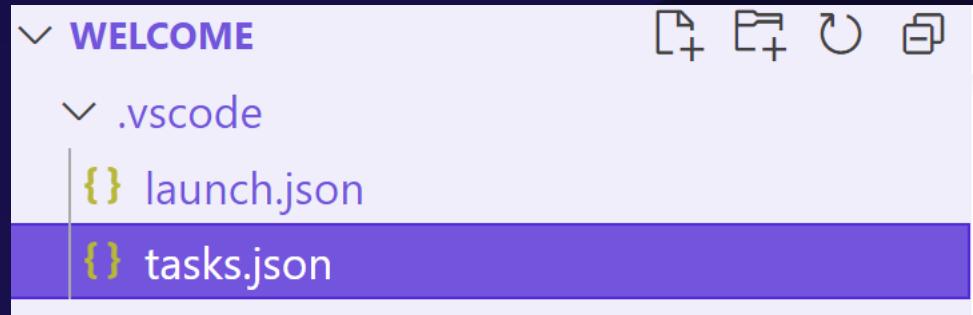
To complete the run experience the app would need to be compile.

preLaunchTasks enable this

In this case it's "publish"

```
{
  "name": ".NET Remote Launch - Raspberry Pi",
  "type": "coreclr",
  "request": "launch",
  "preLaunchTask": "publish",
  "program": "dotnet",
  "args": ["./Welcome.dll"],
  "cwd": "/home/pi/welcome",
  "stopAtEntry": false,
  "console": "internalConsole",
  "pipeTransport": {
    "pipeCwd": "${workspaceRoot}",
    "pipeProgram": "c:\\\\Program Files\\\\PuTTY\\\\plink.exe",
    "pipeArgs": [
      "-pw",
      "${input:PiPassword}",
      "${input:PiHostName}"
    ],
    "debuggerPath": "sudo ~/vsdbg/vsdbg",
  }
}
```

Visual Studio Code



Tasks.json allows the setting up one or many tasks to executed. These tasks can also be configured with task dependencies.

```
{
  "label": "build",
  "command": "dotnet",
  "type": "shell",
  "args": [
    "build",
    // Ask dotnet build to generate full paths for file names.
    "/property:GenerateFullPaths=true",
    // Do not generate summary otherwise it leads to duplicate errors in Problems panel
    "/consoleloggerparameters:NoSummary"
  ],
  "group": "build",
  "presentation": {
    "reveal": "silent"
  },
  "problemMatcher": "$msCompile"
},
{
  "label": "publish",
  "type": "shell",
  "dependsOn": "build",
  "presentation": {
    "reveal": "always",
    "panel": "new"
  },
  "options": {
    "cwd": "${workspaceFolder}"
  },
  "windows": {
    "command": "${cwd}\\\\netpublish.bat",
    "args": ["${input:WinscpSite}"]
  }
},
```

Visual Studio Code

Publish task is configured to call a batch file, "netpublish.bat".

This batch file ensures the built app is built and published as an arm binary.

Winscp synchronizes the binaries between the local built code and the remote Raspberry Pi.

```
netpublish.bat
1 SET currentPath=%cd%
2 ECHO %currentPath%
3 SET buildPath=%currentPath%\bin\Debug\net6.0\linux-arm\publish
4 echo %buildPath%
5 SET command=synchronize remote %buildPath% /home/pi/welcome
6 echo command %command%
7
8 dotnet publish -r linux-arm /p:ShowLinkerSizeComparison=true --self-contained
9 |winscp %1 /command "%command%" "exit"
10
```

Visual Studio Code

Publish task has a dependency of the **build** task.

```
{
  "label": "build",
  "command": "dotnet",
  "type": "shell",
  "args": [
    "build",
    // Ask dotnet build to generate full paths for file names.
    "/property:GenerateFullPaths=true",
    // Do not generate summary otherwise it leads to duplicate errors in Problems panel
    "/consoleloggerparameters:NoSummary"
  ],
  "group": "build",
  "presentation": {
    "reveal": "silent"
  },
  "problemMatcher": "$msCompile"
},
{
  "label": "publish",
  "type": "shell",
  "dependsOn": "build",
  "presentation": {
    "reveal": "always",
    "panel": "new"
  },
  "options": {
    "cwd": "${workspaceFolder}"
  },
  "windows": {
    "command": "${cwd}\\netpublish.bat",
    "args": ["${input:WinscpSite}"]
  },
  "osx": {
    "command": "${cwd}/netpublish.sh",
    "args": ["${input:WinscpSite}"]
  }
}
```

Visual Studio Code

Launch.json and **tasks.json**
supports parameters

Parameters

Pi HostName

Pi Password

WinScp Site Name

Configurable via **.env** file

```
"inputs": [
  {
    "id": "PiHostName",
    "type": "command",
    "command": "extension.commandvariable.file.content",
    "args": {
      "fileName": "${workspaceFolder}/.env",
      "key": "PiHostName",
      "default": ""
    }
  },
  {
    "id": "PiPassword",
    "type": "command",
    "command": "extension.commandvariable.file.content",
    "args": {
      "fileName": "${workspaceFolder}/.env",
      "key": "PiPassword",
      "default": ""
    }
  }
]
```

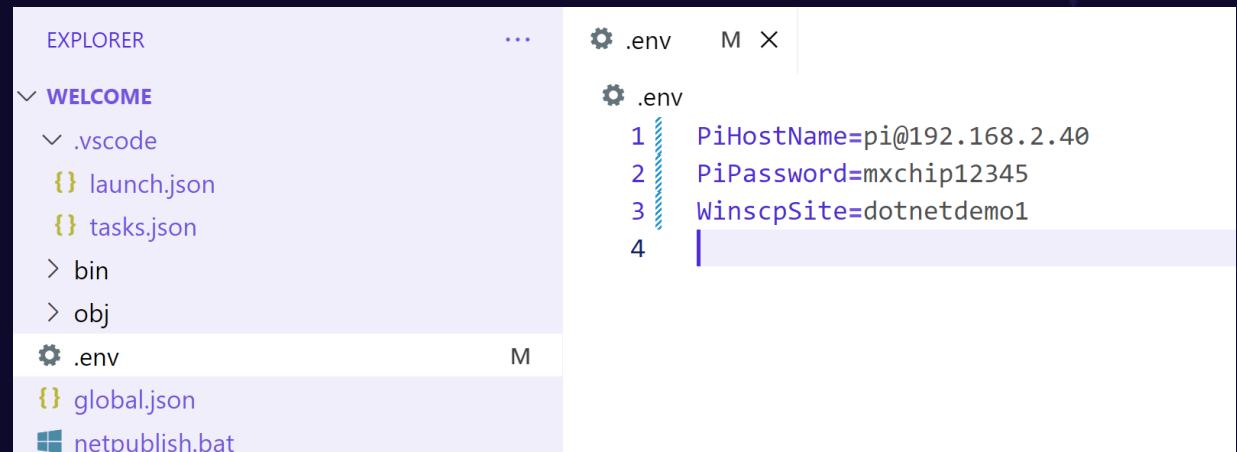
Visual Studio Code

Reuse these preconfigured
Launch.json and **tasks.json** in all
your Raspberry Pi projects to easily
automate remote deployment and
debugging

Just edit the **.env** file and code

Try it out!!

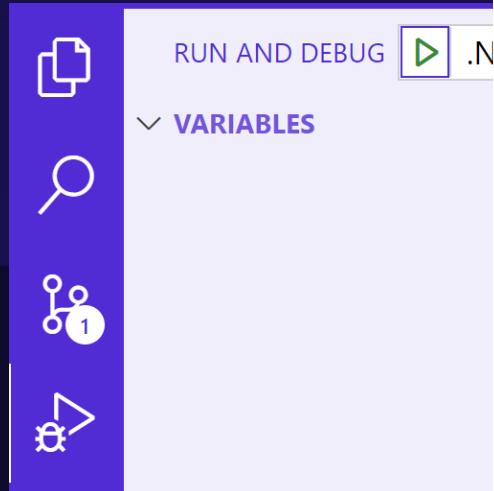
Edit the **.env** file and add your
Pi's details.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar is open, displaying a tree view with 'WELCOME' expanded, showing '.vscode', 'launch.json', 'tasks.json', 'bin', 'obj', '.env', 'global.json', and 'netpublish.bat'. On the right, a code editor window is open with a file named '.env'. The file contains the following environment variables:

```
PiHostName=pi@192.168.2.40
PiPassword=mxchip12345
WinscpSite=dotnetdemo1
```

Visual



... Filter (e.g. text, !exclude)

2

```
.~~. .~~.  
' . \ ' ' / .'  
.~ .~~~...~.  
: .~.'~'.~. :  
~ ( ) ( ) ~  
( : '~~'.~.'~' : )  
~ .~ ( ) ~. ~  
( : '~~' : ) Raspberry Pi  
'~ .~~~. ~'  
'~'
```

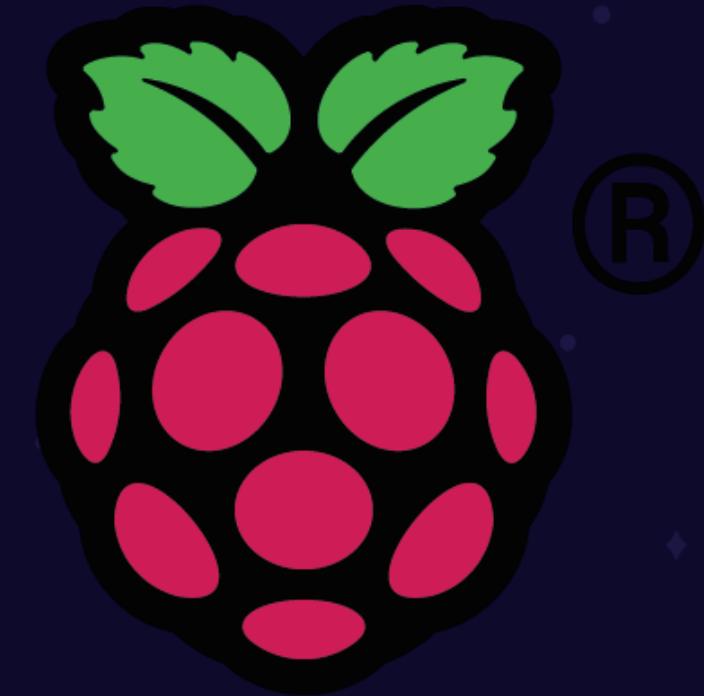
This is a .NET app.

2

2

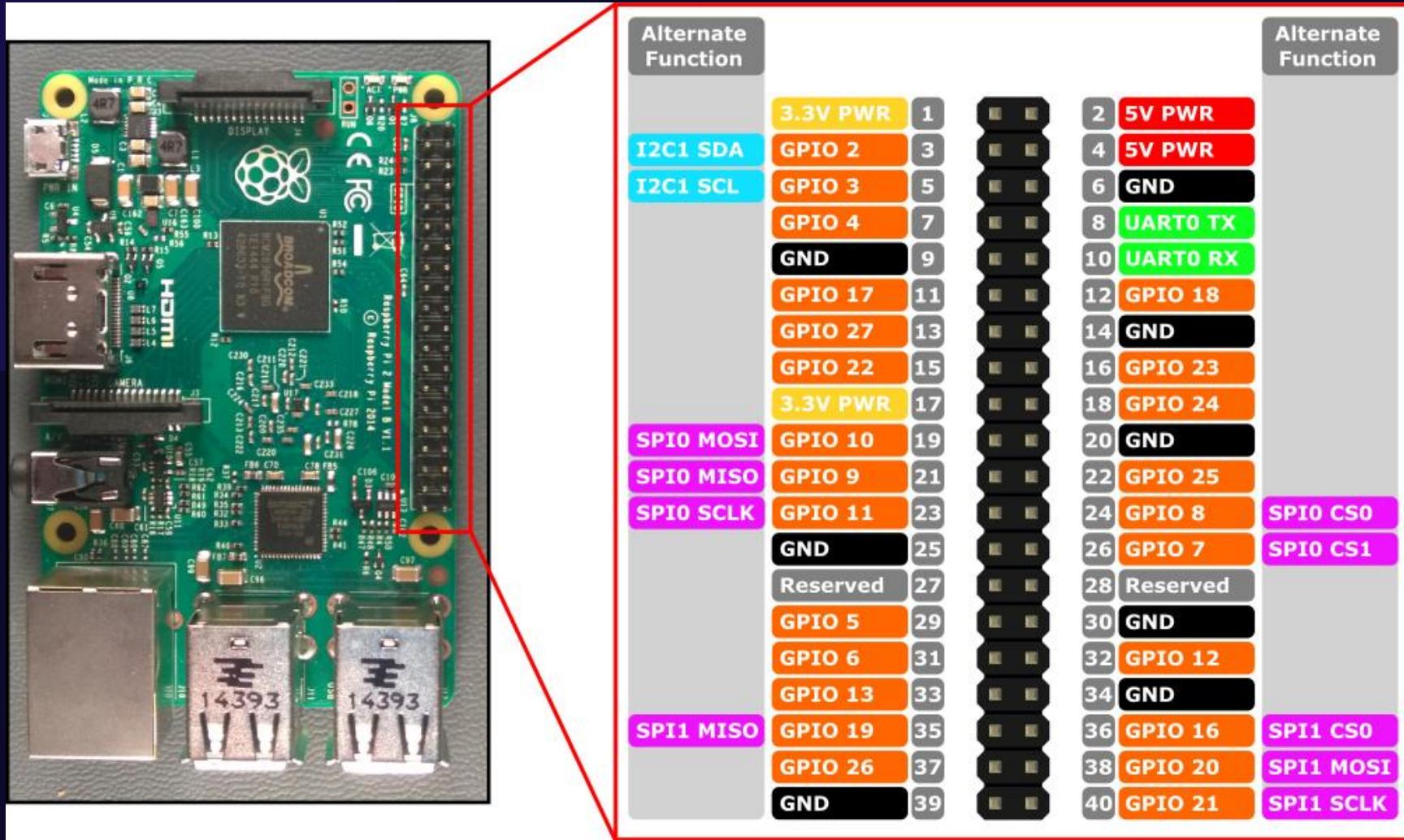
the .NET
erry Pi

Module 2



Getting started with Hardware and Sensors connected to a
Raspberry Pi with .NET

Interfacing to the Raspberry Pi



Interfacing to the Raspberry Pi

GPIO - General-purpose input/output

Switching things on, switching things off

Analog

No just 1 (on) and 0 (off)

Variable Input and Output voltage.

Raspberry Pi has no Analog pins. Requires an Analog to Digital Converter

I2C

Serial communication Bus

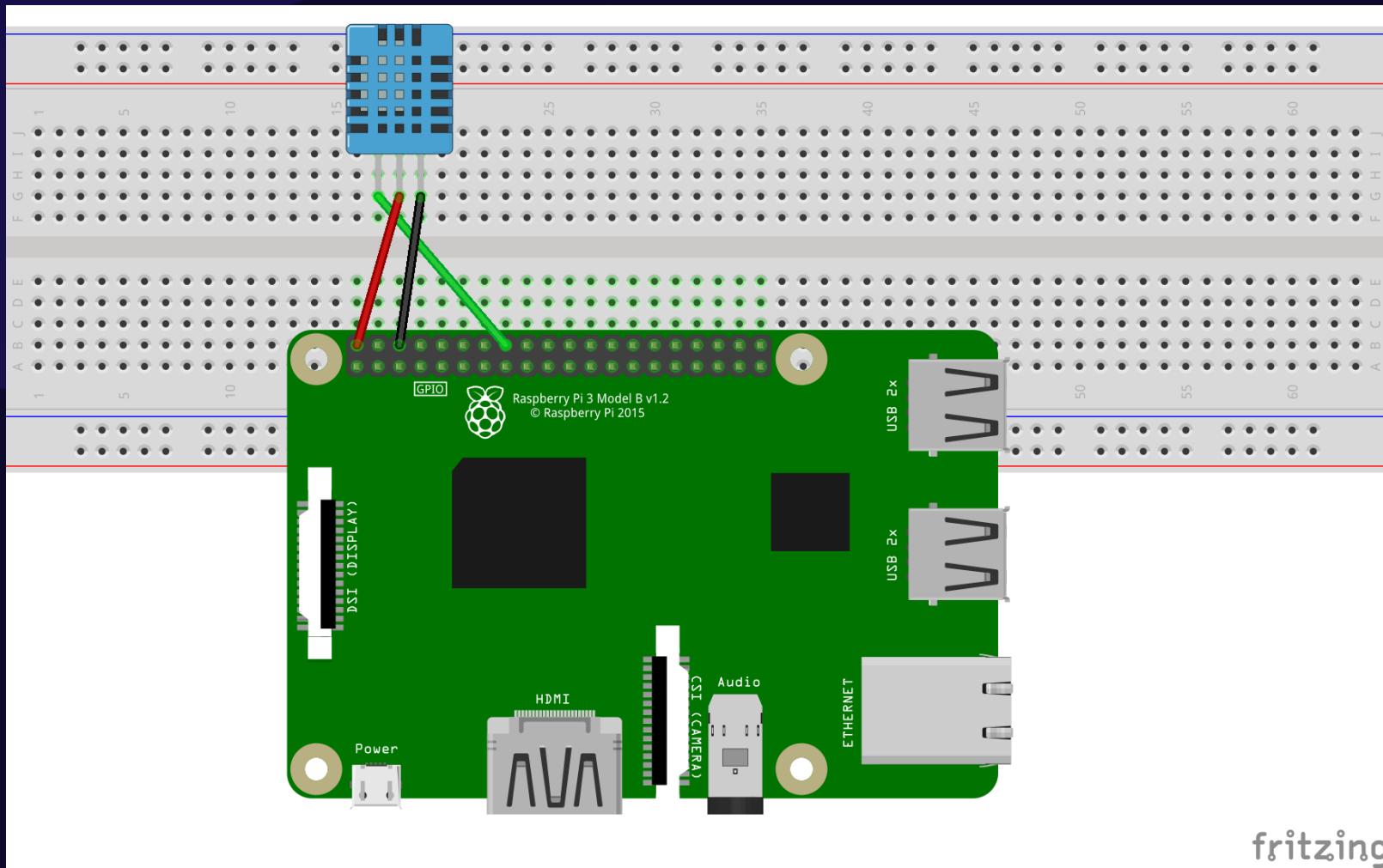
Handy as it's usually 2 wires for complex data communication

SPI (Serial Peripheral Interface)

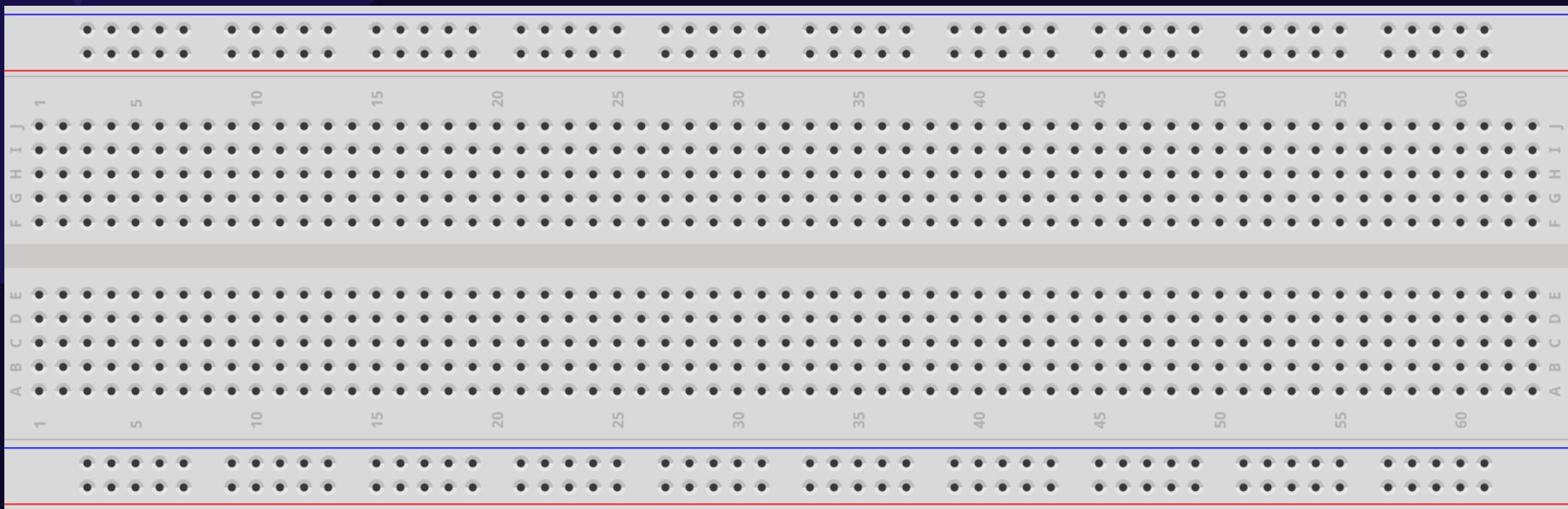
Serial communication bus

Usually 3 wire communication

Let's Build an Environment Monitor



The breadboard



The breadboard

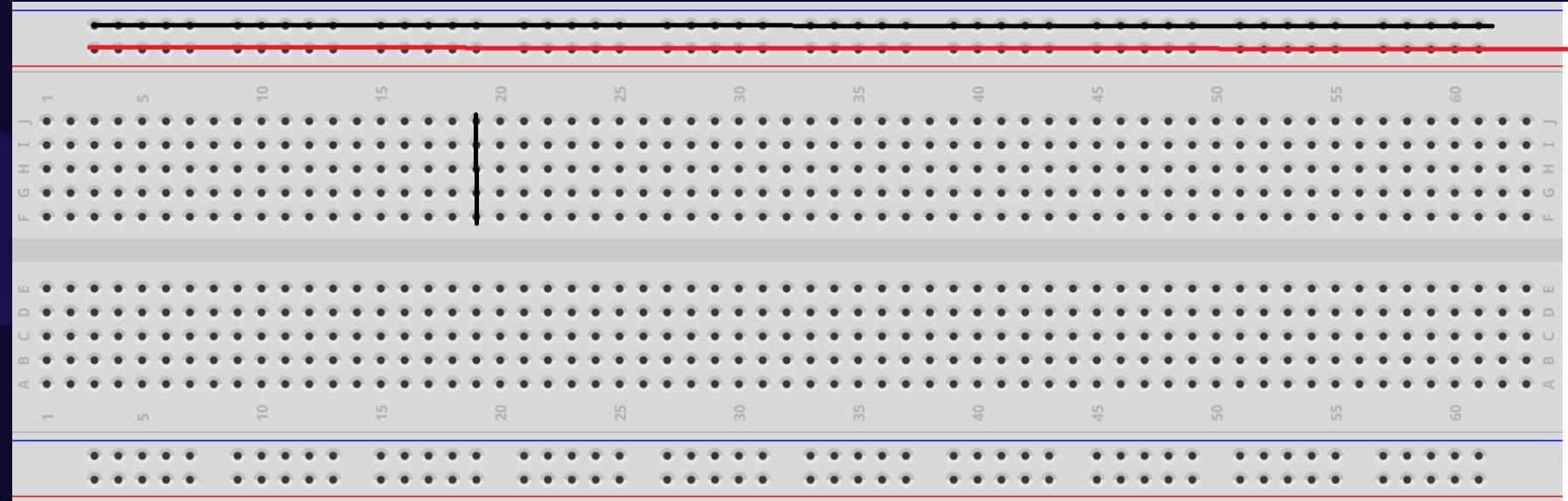
Pins Connected Horizontally

Pins connected vertically

Divider (splits board electrically)

Pins connected vertically

Pins Connected Horizontally

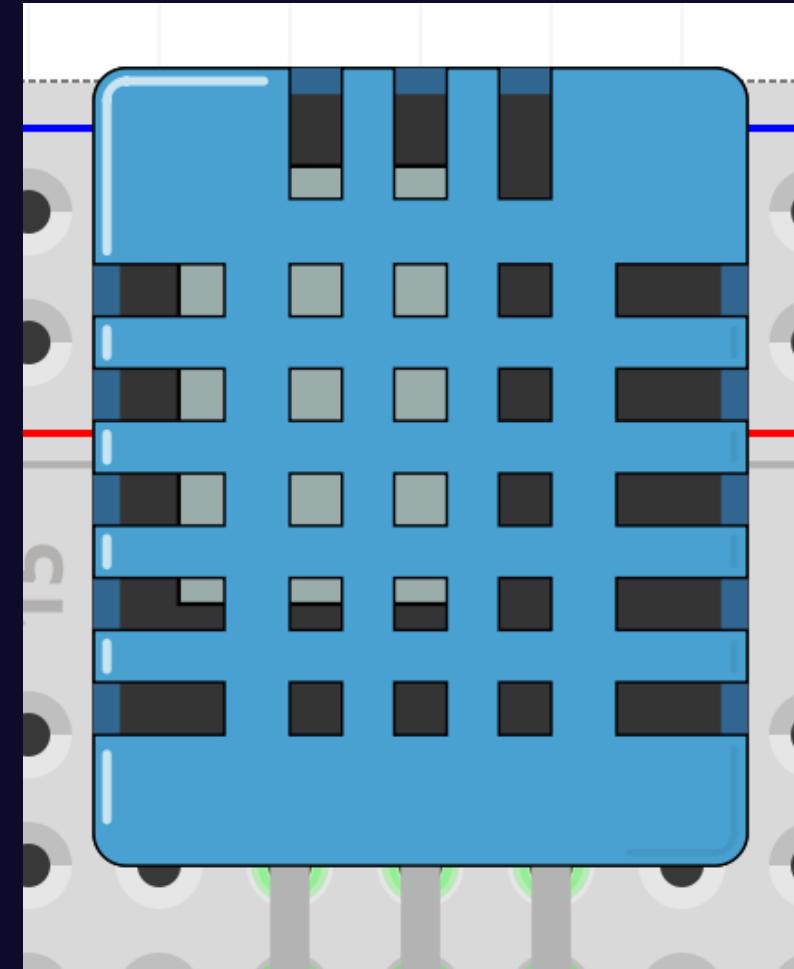


Used only for experimenting and prototyping

DHT 11

Temperature and humidity
sensor

Serial Data



DHT 11

- Pin 2 - Vcc

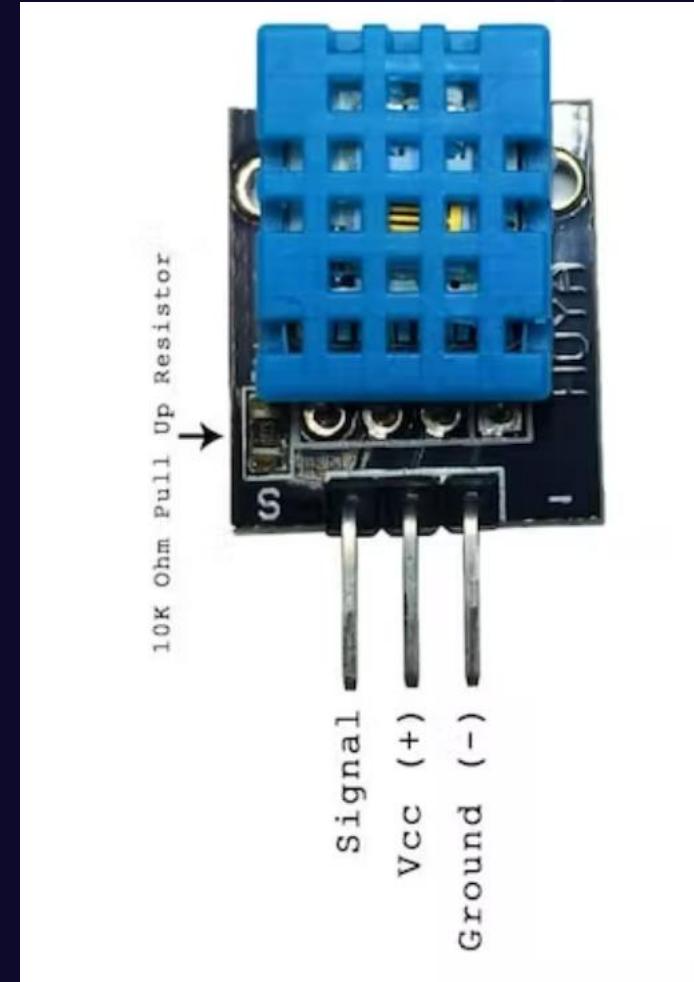
Power supply 3.5V to 5.5V

Pin 1 - Data

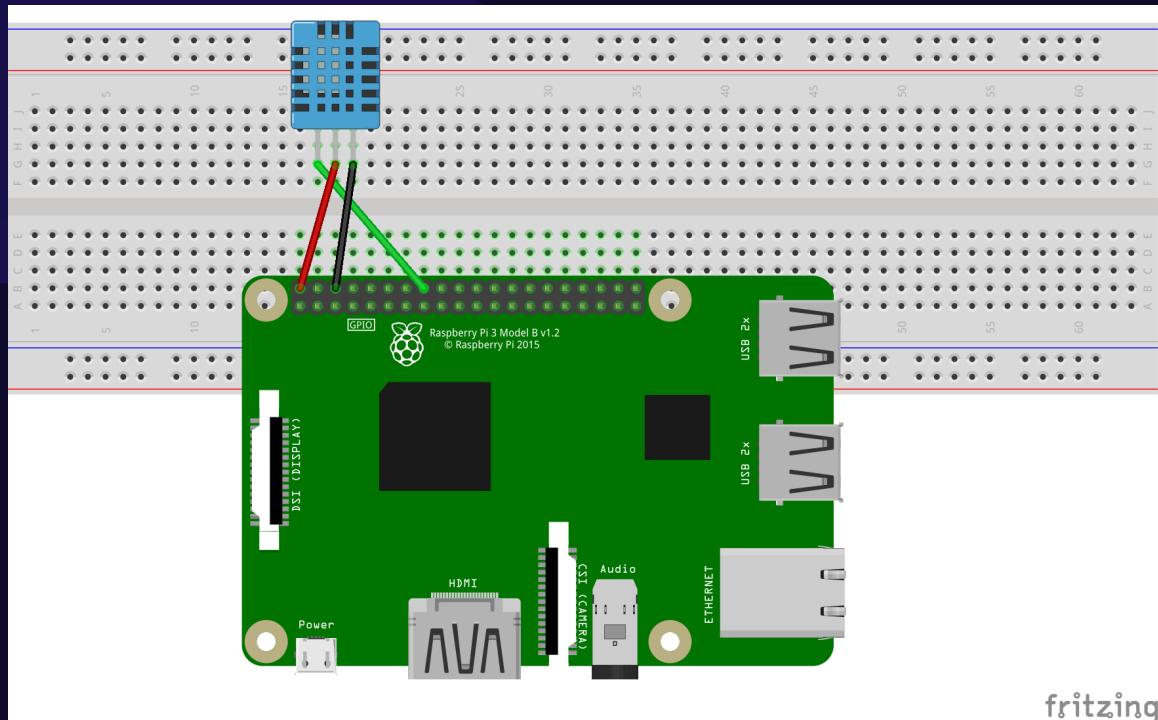
Outputs both Temperature and Humidity through serial Data

Pin 3 - Ground

Connected to the ground of the circuit



Try it out!



Using the component pack, build the circuit using your Raspberry Pi, breadboard, DHT11 and wires.

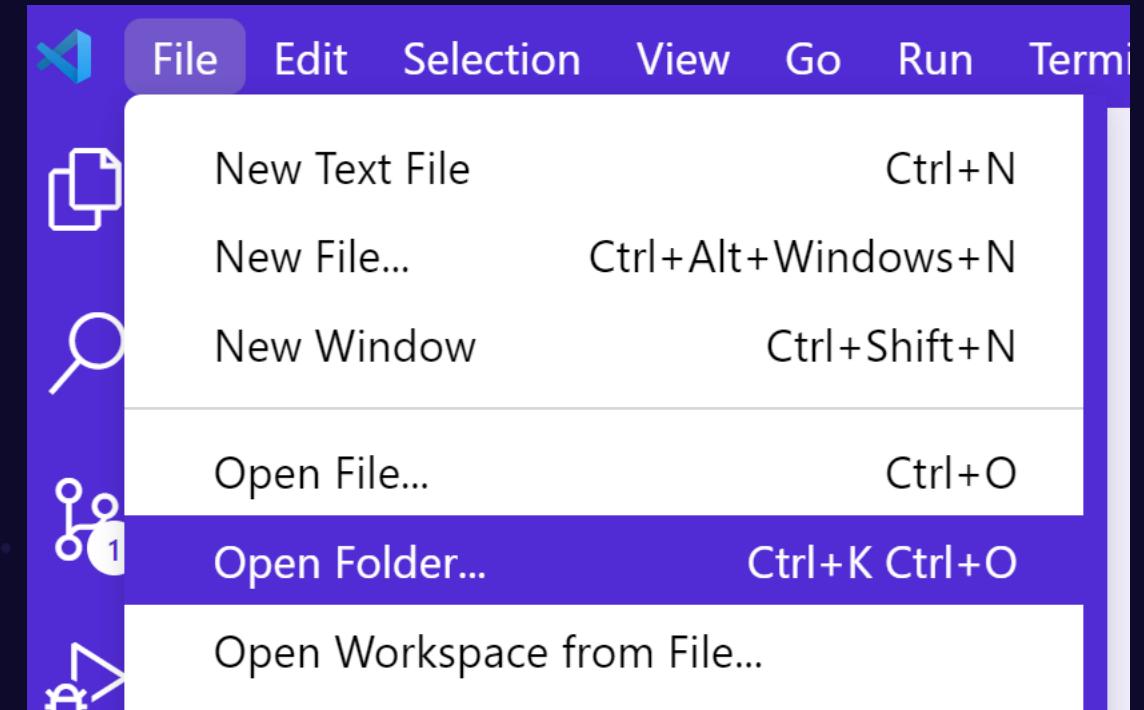
5V from the Raspberry Pi to the middle pin of the DHT 11 Sensor (VCC)

Ground from the Raspberry Pi to the right pin of the DHT Sensor (Ground)

GPIO Pin 23 from the Raspberry Pi to the left pin on the DHT Sensor (Data / Signal)

Try it out!

In VS Code open the folder
SensorStart from the code cloned
from Github

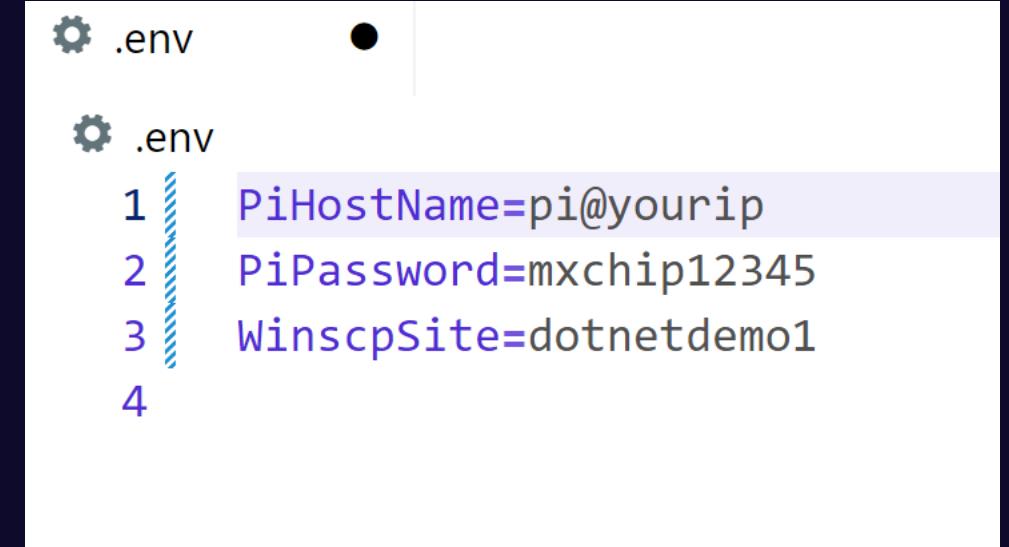


Try it out!

The SensorStart is a starter project
to write the code to read the
Temperature and Humidity

Don't forget to edit the .env file
with the details of your Raspberry
Pi

Test to see if it deploys before you
start coding



```
.env
.
.
.
.env
1 PiHostName=pi@yourip
2 PiPassword=mxchip12345
3 WinscpSite=dotnetdemo1
4
```

Try it out!

The **IoT.Device.Bindings** package has already been included for you

Definition

Namespace: `IoT.Device.DHTxx`

Assembly: `IoT.Device.Bindings.dll`

Package: `IoT.Device.Bindings v2.2.0`

Temperature and Humidity Sensor DHT11

<https://learn.microsoft.com/en-us/dotnet/api/iot.device.dhtxx.dht11?view=iot-dotnet-latest>

```
<PackageReference  
Include="IoT.Device.Bindings"  
Version="2.2.0" />
```

Try it out!

Coding Challenge

Read the Temperature and Humidity repetitively every 5 seconds and display it on the Console (Console.WriteLine).

<https://learn.microsoft.com/en-us/dotnet/api/iot.device.dhtxx.dht11?view=iot-dotnet-latest>

Hints

To create the Dht11 class

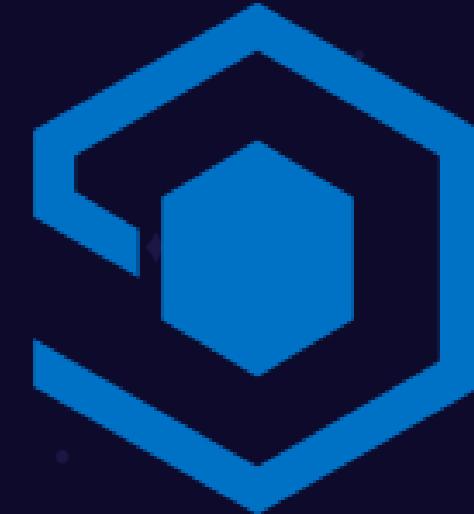
```
Dht11 sensor = new Dht11(23);
```

To attempt to read the Temperature

```
sensor.TryReadTemperature(out  
lastTempReading)
```

```
<PackageReference  
Include="Iot.Device.Bindings"  
Version="2.2.0" />
```

Module 3



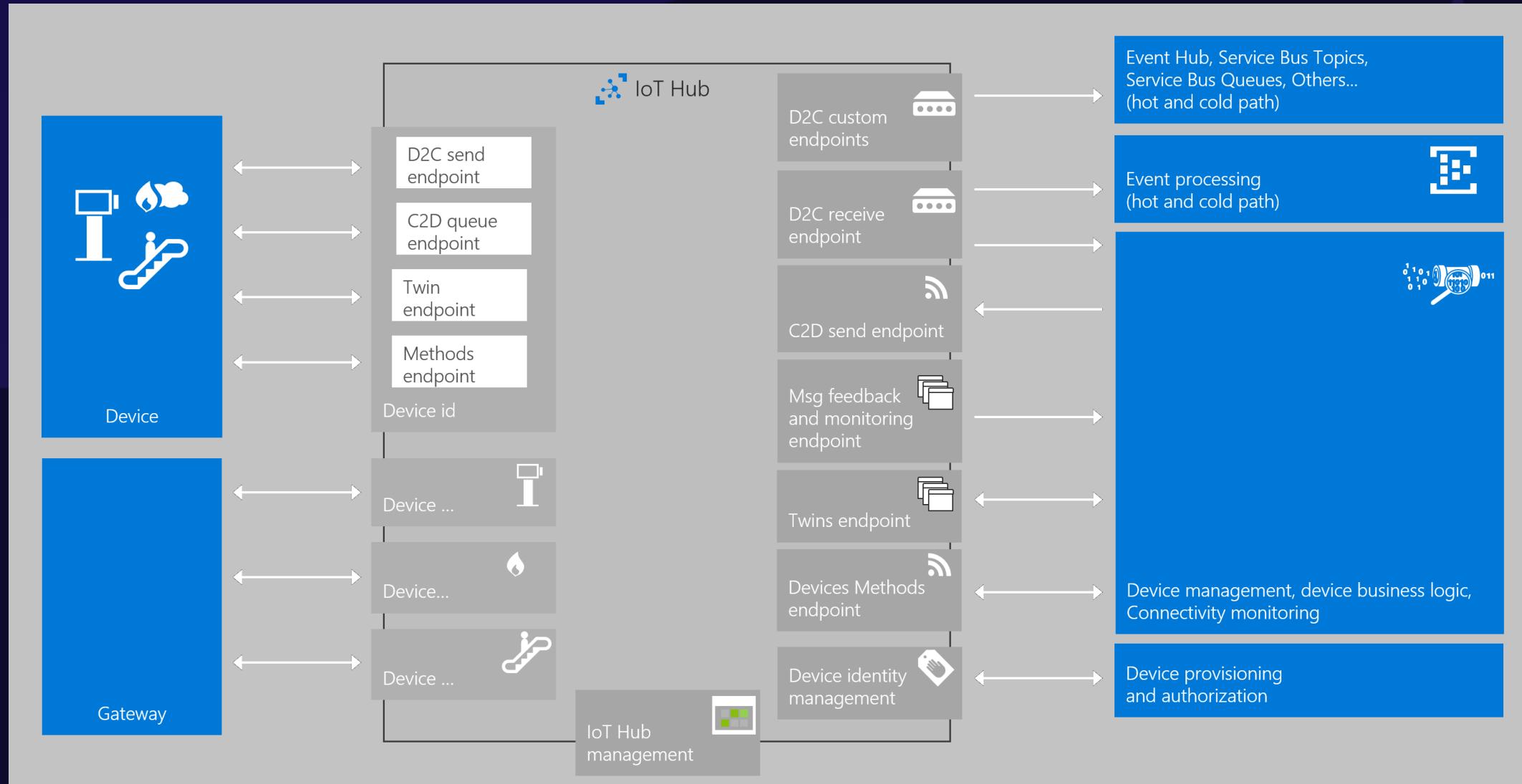
Azure IoT Central

Getting Started with Azure IoT with Azure IoT Central

Azure IoT Hubs

- Establish bi-directional communication with billions of IoT devices
- Work with familiar platforms and protocols (standard and custom protocols, including HTTP, Advanced Message Queuing Protocol (AMQP), and MQ Telemetry Transport (MQTT))
- Authenticate per device for security-enhanced IoT solutions
- Device Management

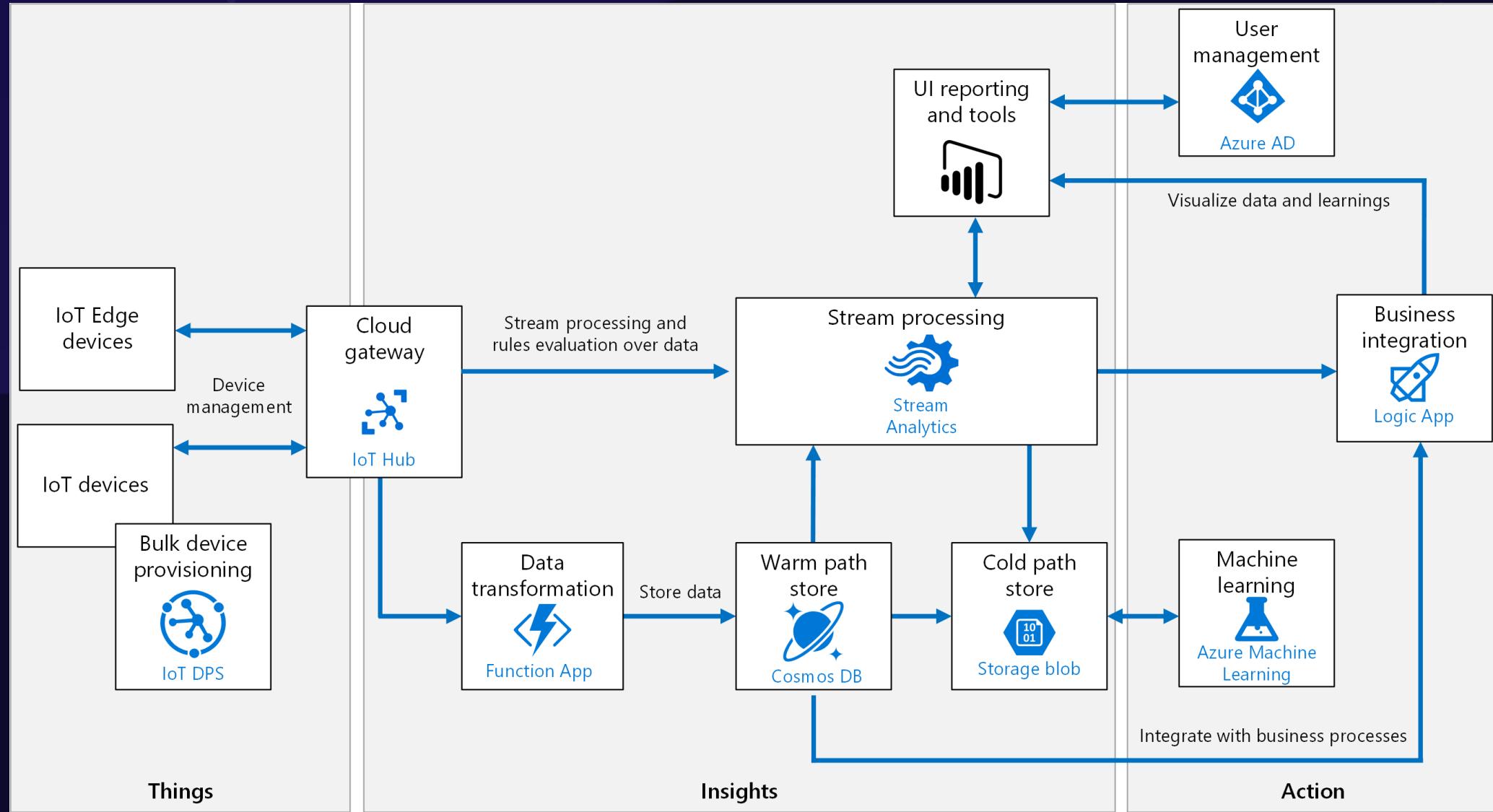
Azure IoT Hubs



Device SDKs platform/OS support

- **Android (Java or Xamarin)**
- **Arduino**
- **Debian Linux (v 7.5+)**
- **ESP8266**
- **Fedora Linux (v 20+)**
- **FreeRTOS**
- **iOS (Xamarin)**
- **mbed OS (v 2.0+)**
- **OpenWRT**
- **Raspbian Linux (v 3.18+)**
- **STM32**
- **TI RTOS**
- **Ubilinux (v3.0+)**
- **Ubuntu Linux (v 14.04+)**
- **Windows Desktop (7, 8, 10)**
- **Windows IoT Core (v 10)**
- **Windows Server (v 2012 R2+)**
- **Yocto Linux (v 2.1+)**
- **Azure Sphere**
- **... and more**

Azure IoT Reference Architecture



IoT Solutions - IoT Central



Azure IoT Central

Experience the simplicity of SaaS for IoT, with no cloud expertise required



Fast and easy

- Start in minutes and create a finished solution in hours
- Build without any cloud development expertise



Scalable and secure

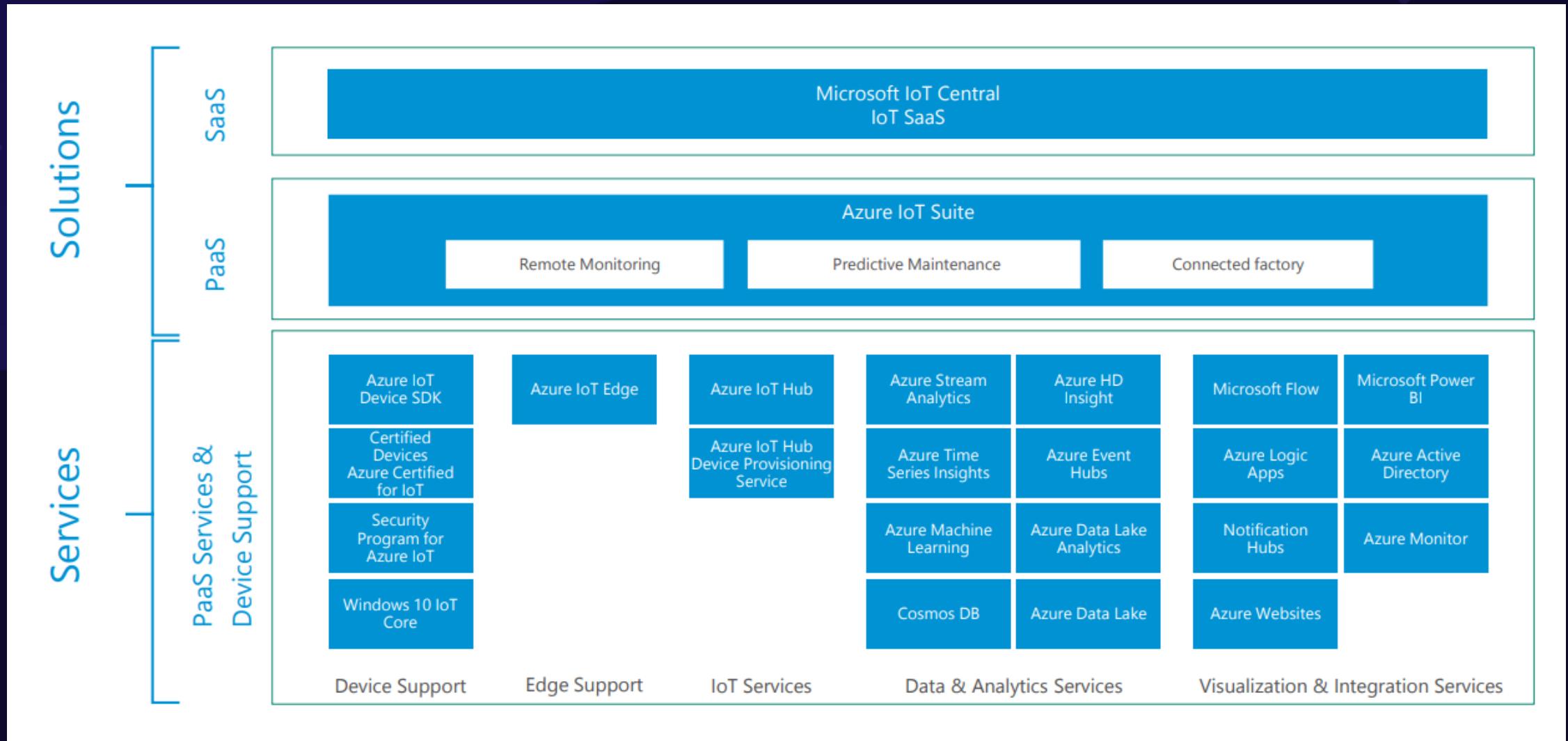
- Connect your devices at any scale without worrying about infrastructure
- Get best-in-class security



Enterprise grade

- Built on proven enterprise-grade Azure services
- Full integration into your existing business systems and processes

IoT Central



IoT Central - Device Templates

- Model a Device without writing code
- Model Telemetry
- Model Device Commands
- Model Device Properties
- Simulate Devices
- Versionable Devices
- Supports IoT Plug 'n Play

The screenshot shows the Azure IoT Central interface for managing device templates. The left sidebar has a navigation menu with items like Dashboard, Devices, Device groups, Rules, Analytics, Jobs, App settings, and Device templates (which is selected). The main content area displays the details for the 'Air-Quality-Monitor' template under the 'Pimoroni Enviro+' application. The 'Summary' section lists various telemetry interfaces: Temperature, pressure, humidity, light, oxidation, nh3, and redu. Each entry includes the display name, name, capability type (Telemetry), and interface type (Interface). The top right corner shows a 'Published' status.

Display name	Name	Capability type	Interface
Temperature	temperature	Telemetry	Interface
Pressure	pressure	Telemetry	Interface
Humidity	humidity	Telemetry	Interface
Light	light	Telemetry	Interface
Oxidation	oxidation	Telemetry	Interface
Nh3	nh3	Telemetry	Interface
Redu	redu	Telemetry	Interface

Device Templates - Capabilities

- Model the device sensors or commands
- Model types and units of telemetry item
- Versionable Interface

The screenshot shows the 'Interface' tab of a device template named 'Air-Quality-Monitor'. The left sidebar includes 'Pimoroni Enviro+' (selected), 'Interface' (selected), 'Cloud properties', 'Customize', 'Views' (with 'Overview' and 'About' options), and a 'Published' status indicator. The main area is titled 'Capabilities' and contains a table for defining data types and their properties:

Display name *	Name *	Capability type *	Semantic type
Temperature	temperature	Telemetry	Temperature
Pressure	pressure	Telemetry	None
Humidity	humidity	Telemetry	None
Light	light	Telemetry	None
Oxidation	oxidation	Telemetry	None
Nh3	nh3	Telemetry	None
Redu	redu	Telemetry	None

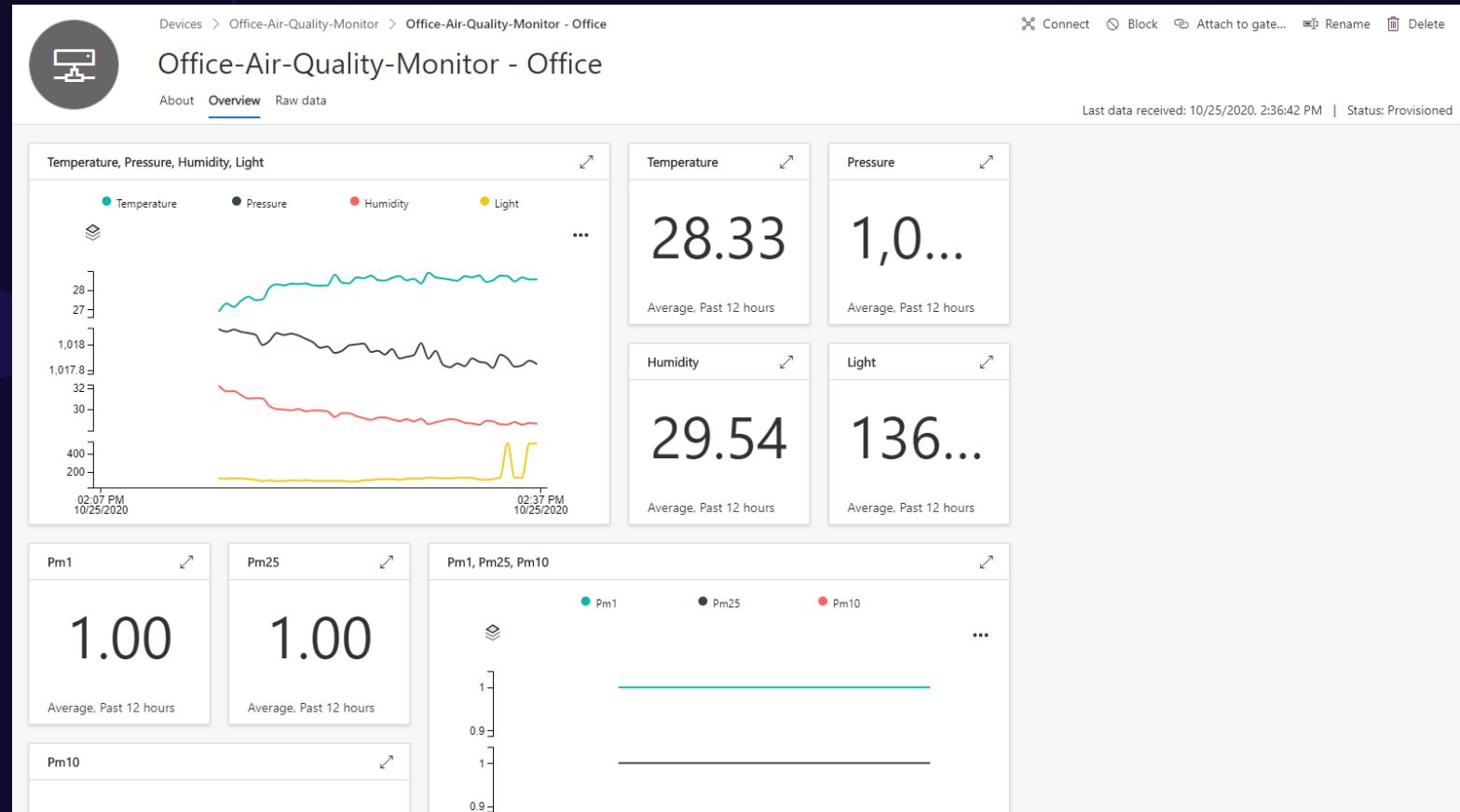
Commands

- Model Commands to be invoked on Device
- Model Request / input parameters
- Model Response / output parameters
- Versionable Interface

The screenshot shows the Azure Device Template interface for a "Home-Office-Gateway" device. The top navigation bar includes "Version", "Manage test device", "Publish", "Rename", and "Delete". Below the navigation is the device name "Home-Office-Gateway" and a note indicating it was updated and published 5 days ago. The left sidebar has sections for "Home-Office-Gateway" (selected), "Interface" (selected), "Cloud properties", and "Customize". The main content area is titled "Interface" and contains fields for defining commands, requests, and responses. For the "Command" section, the display name is "Turn Gateway Device On", the name is "turngatewaydeviceon", the capability type is "Command", and the semantic type is "String". The "Queue if offline" option is set to "Off". For the "Request" section, the display name is "Device Name", the name is "devicename", the schema is "String", and there is a "View" link. The "Response" section has a display name of "Response", a name of "response", a schema of "String", and a "View" link. There are also "Comment" and "Description" fields for each section.

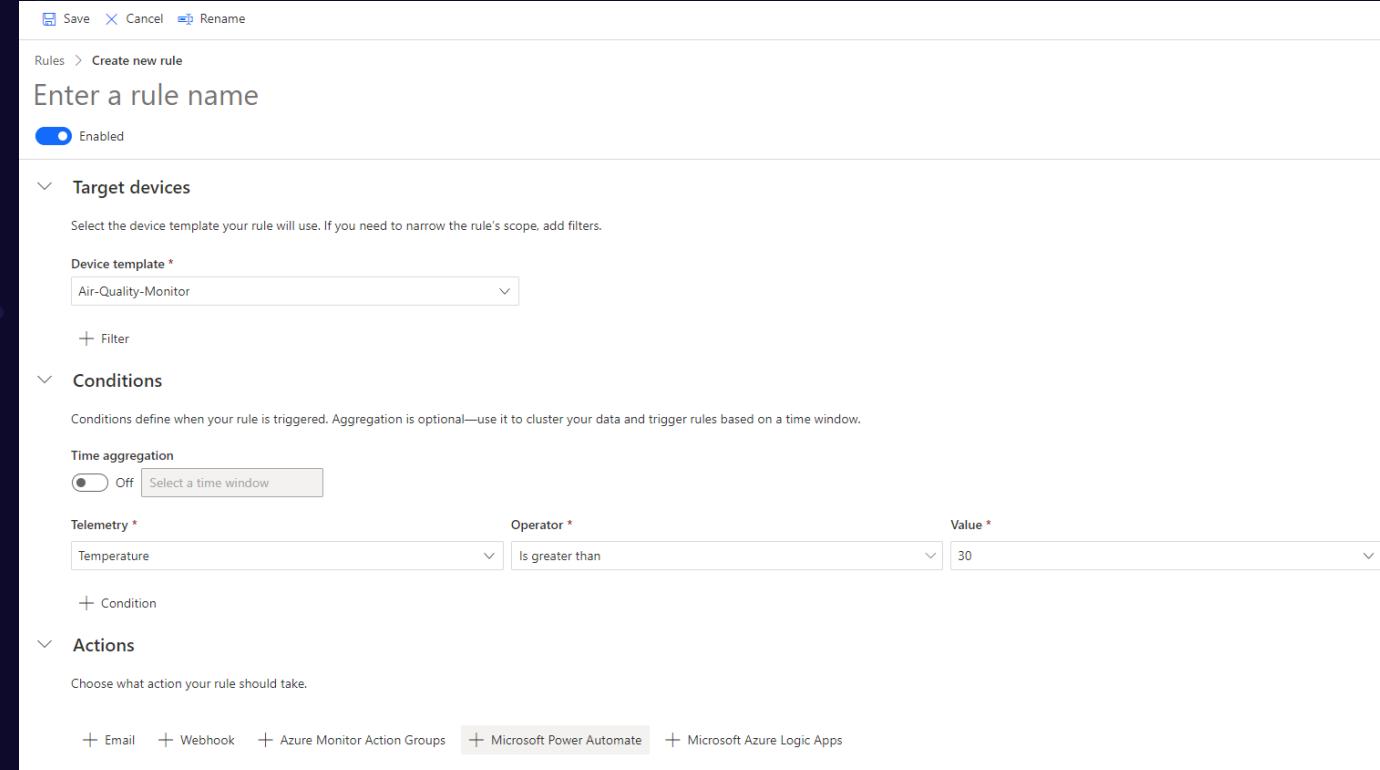
Dashboards

- Device Dashboards
- Customize Dashboards
- Customize Dashboards per user
- Analytics Views



Rules

- Create rules based on conditions of modeled capabilities
- Performs Actions based on conditions met
 - Email
 - Webhook
 - Power Automate
 - Logic Apps
 - Azure Monitor Action Groups

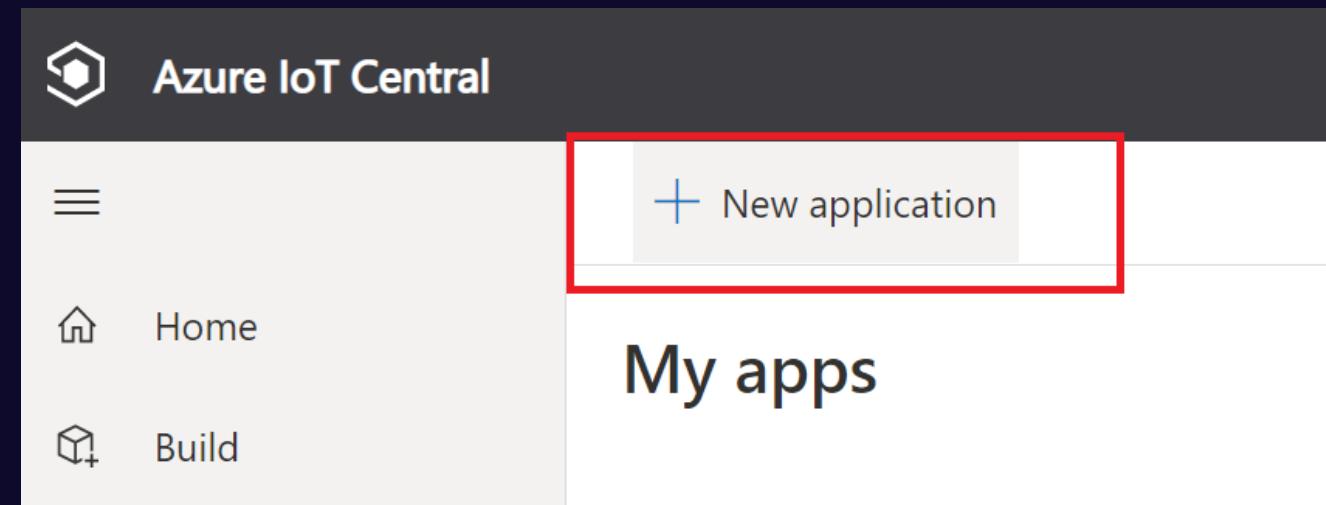


Try it Out!

Log into Azure IoT Central

<https://apps.azureiotcentral.com/>

Create a New Application



This Requires an Azure Subscription!

Try it Out!

Create Custom App

Build your IoT application

Featured



Custom app

Create a custom application to build a unique solution for your business using powerful tools to connect, monitor, and manage your IoT data.

[Create app](#)

[Learn more](#)

This Requires an Azure Subscription!

Try it Out!

Give your Application a Name

Chose the Standard 0 Price Plan (2 devices free)

The screenshot shows the 'Build > New application' interface. On the left, a sidebar has 'Home', 'Build' (which is selected), and 'My apps'. The main area is titled 'New application' with a 'Custom' tab. It says 'Answer a few quick questions and we'll get your app up and running.' Under 'About your app', the 'Application name *' field contains 'IoT Hack Session 1'. The 'URL *' field contains 'iot-hack-session-1' followed by '.azureiotcentral.com'. The 'Application template *' dropdown is set to 'Custom application'. Under 'Pricing plan', 'Standard 0' is selected, described as 'For devices sending a few messages per day' with '2 free devices 400 messages/mo'.

Build > New application

New application Custom

Answer a few quick questions and we'll get your app up and running.

About your app

Application name * ⓘ

IoT Hack Session 1

URL * ⓘ

iot-hack-session-1 .azureiotcentral.com

Application template * ⓘ

Custom application

Pricing plan

Standard 0

For devices sending a few messages per day

2 free devices 400 messages/mo

This Requires an Azure Subscription!

Try it Out!

Choose an Active Azure Subscription

Create the Application

Billing info

Directory * ⓘ

Azure subscription * ⓘ Don't have a subscription? [Create subscription ↗](#)

Microsoft Azure Sponsorship

Location * ⓘ

West Europe

By clicking "Create" you agree to the [Subscription Agreement ↗](#) and [Privacy Statement ↗](#). "Standard" plans require an Azure subscription, and you acknowledge that this service is licensed to you under the terms applicable to your [Azure Subscription ↗](#).

Create Cancel

This Requires an Azure Subscription!

Try it Out!

Create a Device Template

The screenshot shows the Azure IoT Hub interface titled "IoT Hack Session 1". On the left, there's a navigation menu with options: Connect, Devices, Device groups, **Device templates** (which is selected and highlighted with a dashed blue border), Edge manifests, Analyze, and Data explorer. On the right, under the heading "Device templates", there's a message: "IoT Hub and DPS are updating their TLS certificates with a new M...". Below the message is a blue button with a white plus sign and the text "+ New", which is also highlighted with a red box. A search bar is visible at the top right.

This Requires an Azure Subscription!

Try it Out!

Select IoT Device

Device templates > Create new device template

Select type

A device template is like a blueprint. It defines the characteristics and behaviors of devices that connect to your app.

Create a custom device template



IoT device
Import a capability model or build capabilities from scratch.



Azure IoT Edge
Create a template that features Azure IoT Edge and gateway scenarios.

This Requires an Azure Subscription!

Try it Out!

Provide the Template a name

Device templates > Create new device template

Customize

Device template name*

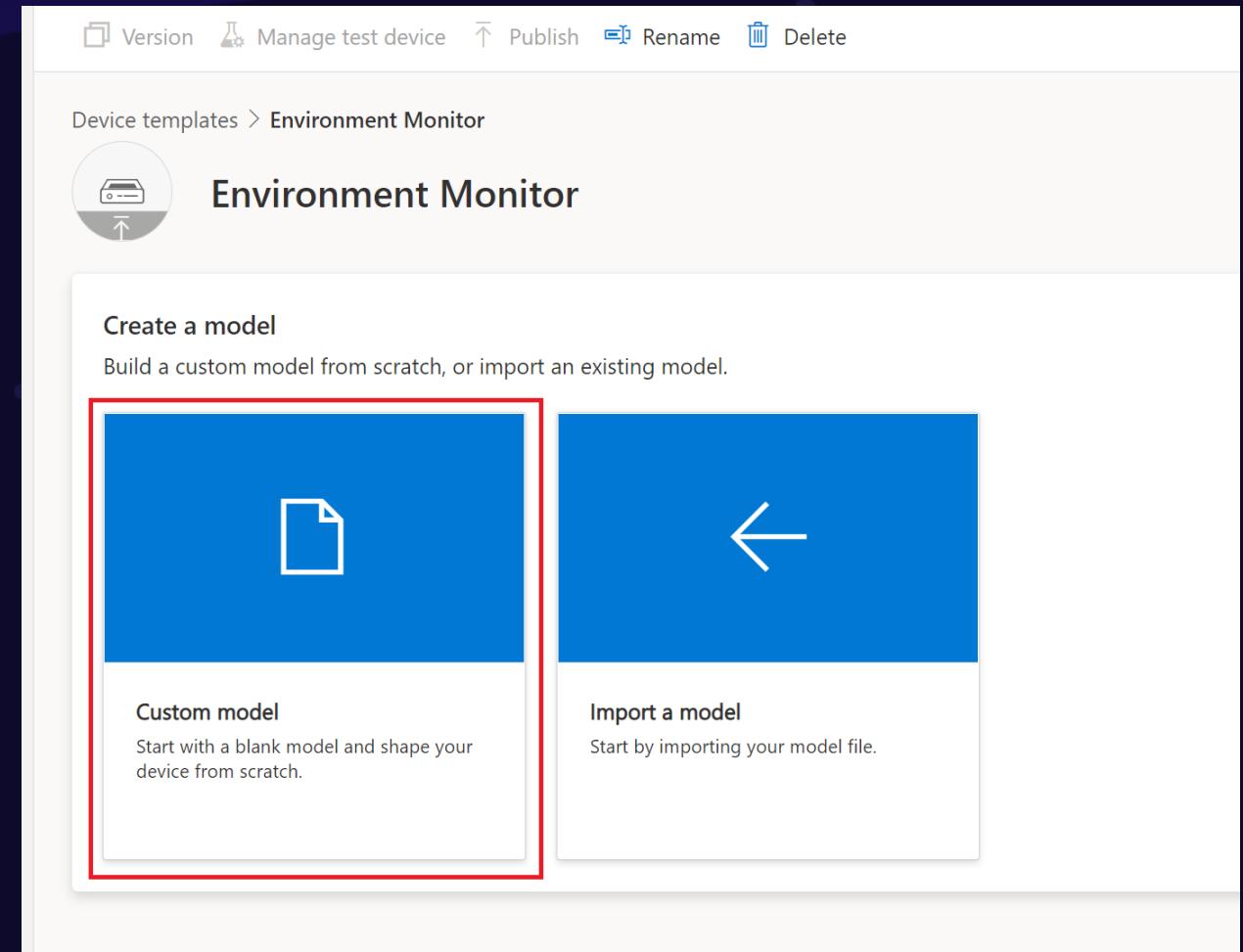
 x

This is a gateway device. [Learn more.](#)

This Requires an Azure Subscription!

Try it Out!

Create a Custom Model for
the Device Template



This Requires an Azure Subscription!

Try it Out!

Add a Capability

Device templates > Environment Monitor > Model > Environment Monitor

Environment Monitor

Application updated: Never Interfaces published: Never

> Environment Monitor **Root** Draft

Add capabilities specific to this device model. [Learn more](#)

Save + Add capability Edit identity Export Delete ...

+ Add capability

This Requires an Azure Subscription!

Try it Out!

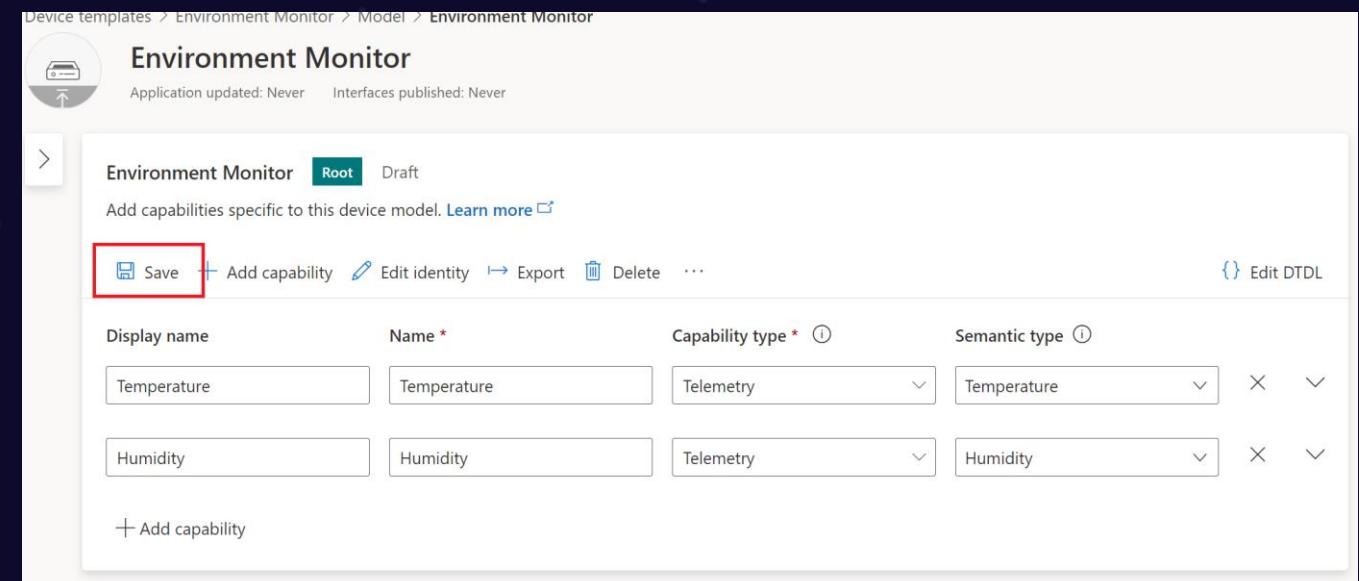
Add a Capability for Temperature

- Semantic Type Temperature

Add a Capability for Humidity

- Semantic Type Humidity

Make sure you Save

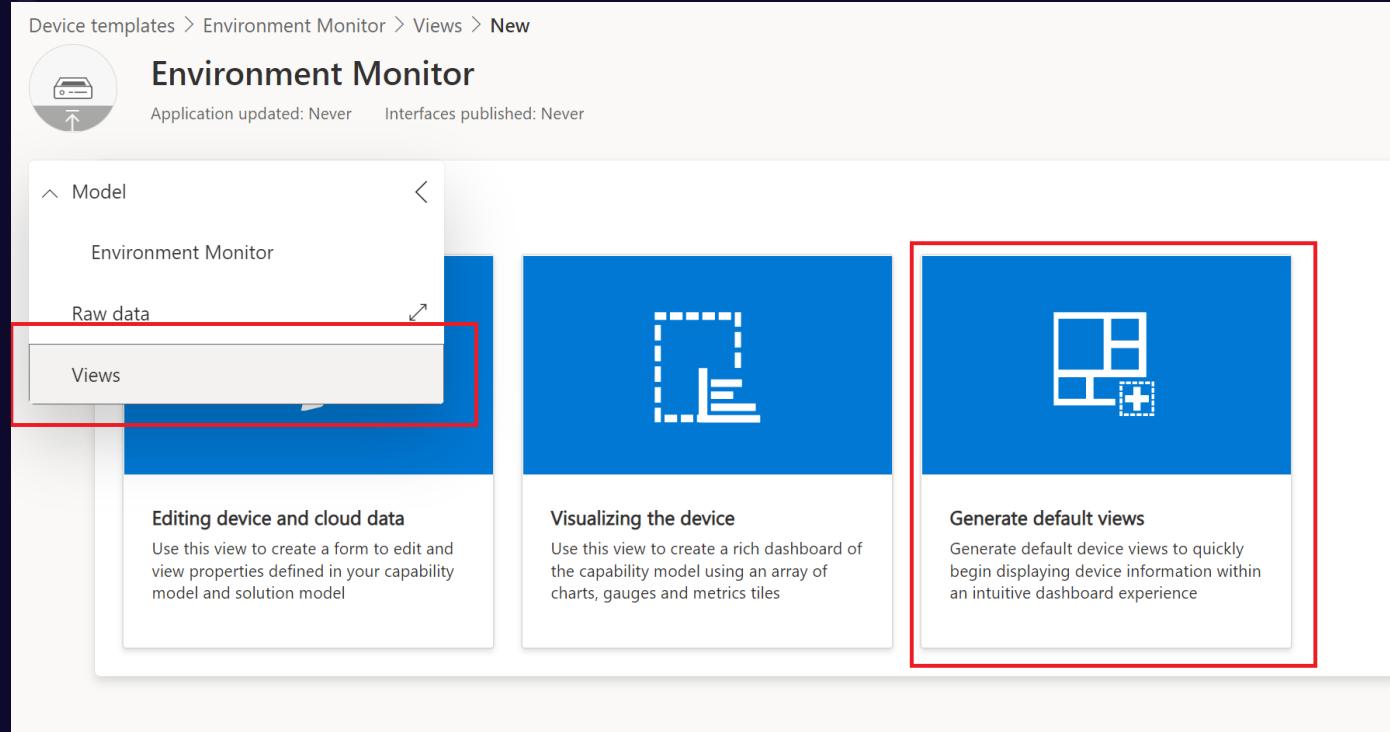


The screenshot shows the Azure Device Template Editor interface. The path in the top left corner is "Device templates > Environment Monitor > Model > Environment Monitor". The main title is "Environment Monitor" with a subtitle "Application updated: Never" and "Interfaces published: Never". Below the title, there's a breadcrumb navigation "Environment Monitor > Root Draft". A message says "Add capabilities specific to this device model. Learn more". A toolbar below has a "Save" button (highlighted with a red box), "Add capability", "Edit identity", "Export", "Delete", and a "..." button. To the right of the toolbar is a "Edit DTDL" link. The main area displays two capability rows. Each row has four columns: "Display name", "Name *", "Capability type *", and "Semantic type". The first row has "Temperature" in all four columns. The second row has "Humidity" in all four columns. At the bottom of the list is a "+ Add capability" button.

This Requires an Azure Subscription!

Try it Out!

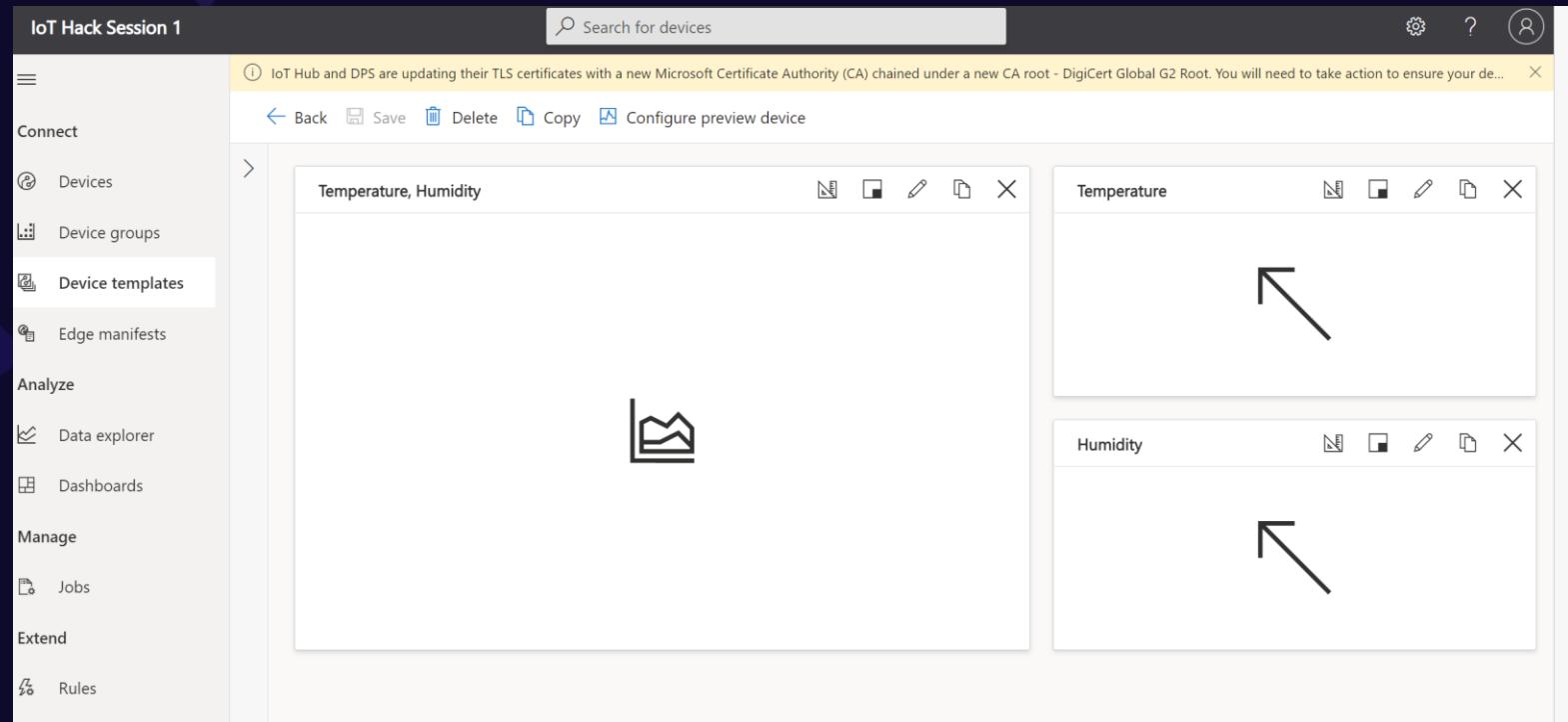
Auto-generate a view / dashboard from the model created.



This Requires an Azure Subscription!

Try it Out!

Generated Views

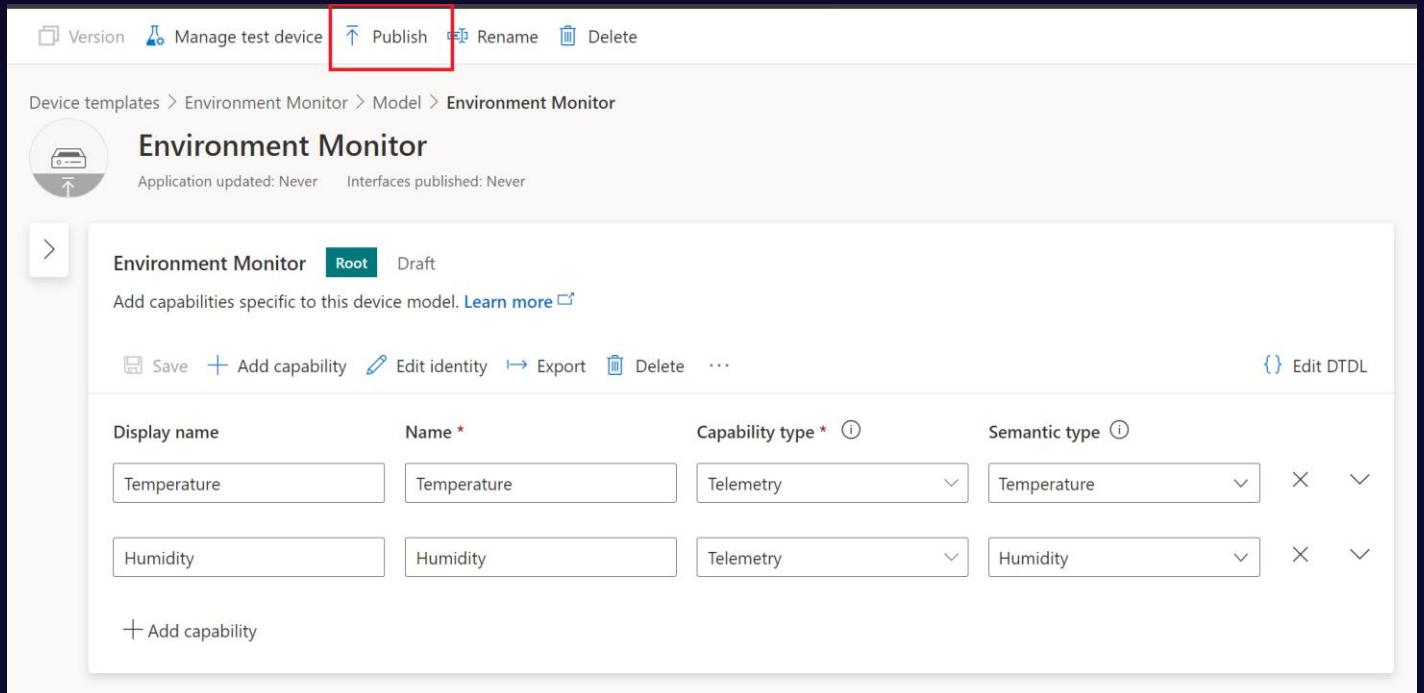


This Requires an Azure Subscription!

Try it Out!

Publish a version of the model

The versioned model can now
be used as a device.

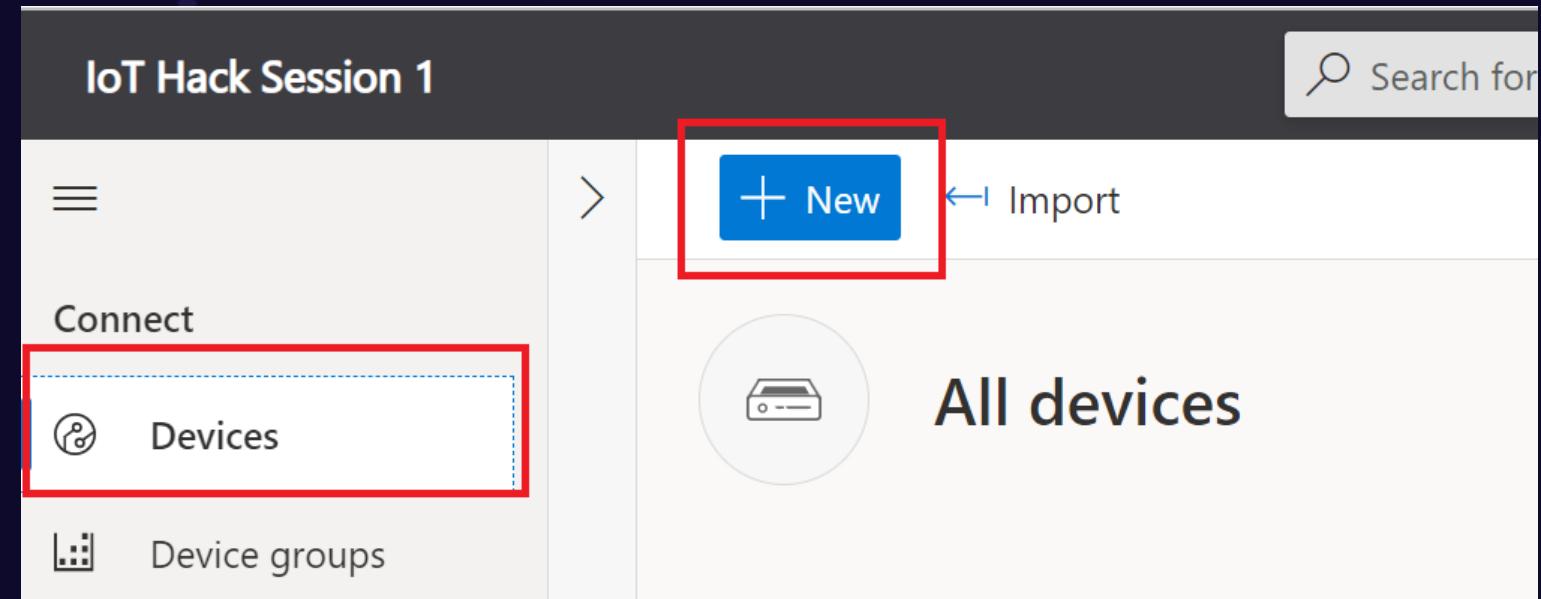


The screenshot shows the Azure Device Template Editor interface. At the top, there's a navigation bar with 'Version', 'Manage test device', 'Publish' (which is highlighted with a red box), 'Rename', and 'Delete'. Below the navigation bar, the path 'Device templates > Environment Monitor > Model > Environment Monitor' is displayed. The main title 'Environment Monitor' is shown with a small icon of a device. Below the title, it says 'Application updated: Never' and 'Interfaces published: Never'. Underneath, there's a section titled 'Environment Monitor' with a 'Root' status and 'Draft' status. It includes a link 'Add capabilities specific to this device model. Learn more'. Below this, there are four rows of configuration fields for 'Temperature' and 'Humidity'. Each row has 'Display name', 'Name *', 'Capability type *', and 'Semantic type' dropdowns. The 'Temperature' row has 'Temperature' in all fields. The 'Humidity' row has 'Humidity' in all fields. There are also 'Edit identity', 'Export', 'Delete', and a 'Save' button. At the bottom, there's a '+ Add capability' button. On the right side of the interface, there's a link '(?) Edit DTDL'.

This Requires an Azure Subscription!

Try it Out!

Now create an instance of a device based on the model created



This Requires an Azure Subscription!

Try it Out!

Give the device a name

Assigned the previously created device template to this instance of the device

Create the Device

Create a new device

To create a new device, select a device template, a name, and a unique ID. [Learn more](#)

Device name * ⓘ
Samsung Lab Monitor

Device ID * ⓘ
v42xge3m6q

Organization * ⓘ
IoT Hack Session 1

Device template * ⓘ
 Environment Monitor

Unassigned

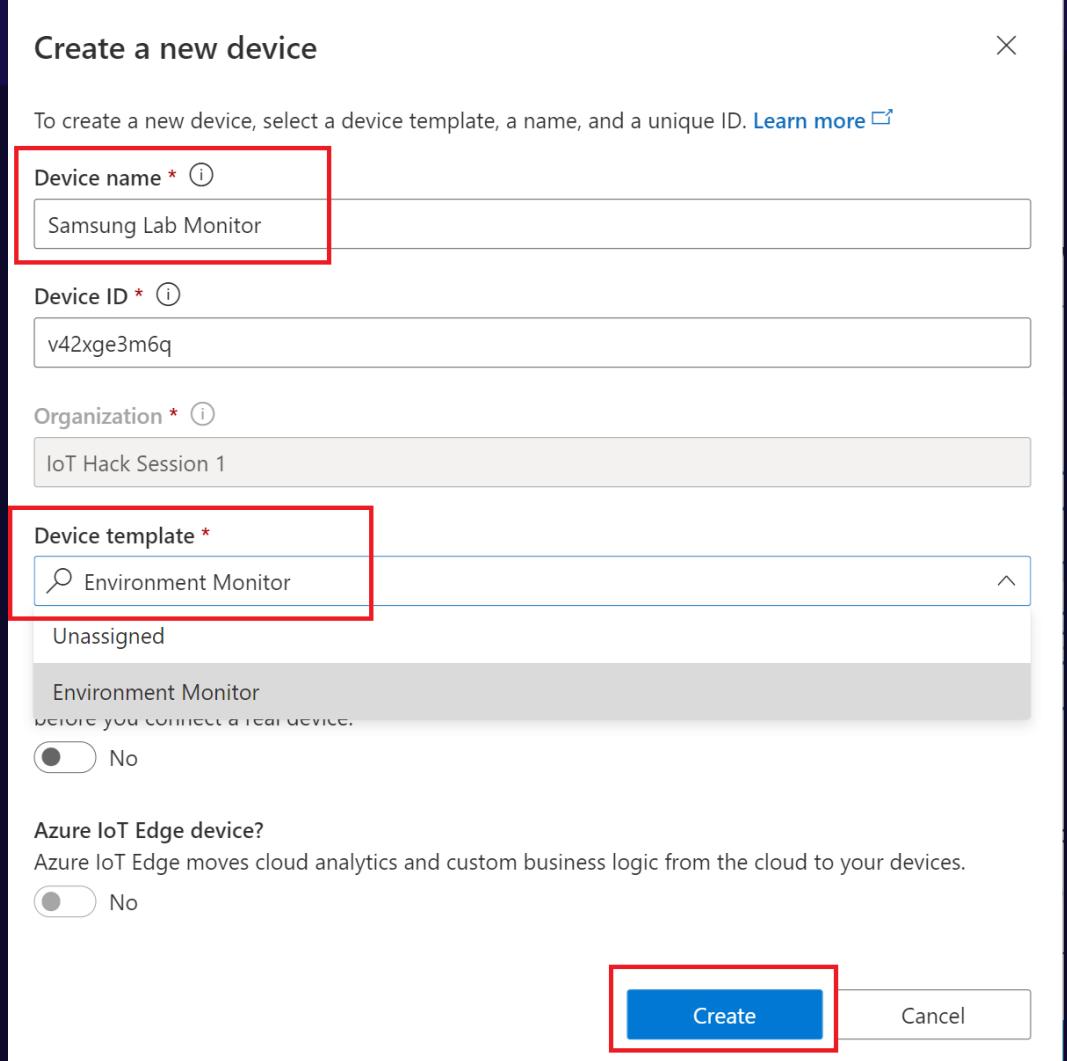
Environment Monitor
before you connect a real device.

No

Azure IoT Edge device?
Azure IoT Edge moves cloud analytics and custom business logic from the cloud to your devices.

No

Create Cancel



This Requires an Azure Subscription!

Try it Out!

The device has been created in Azure IoT Central.

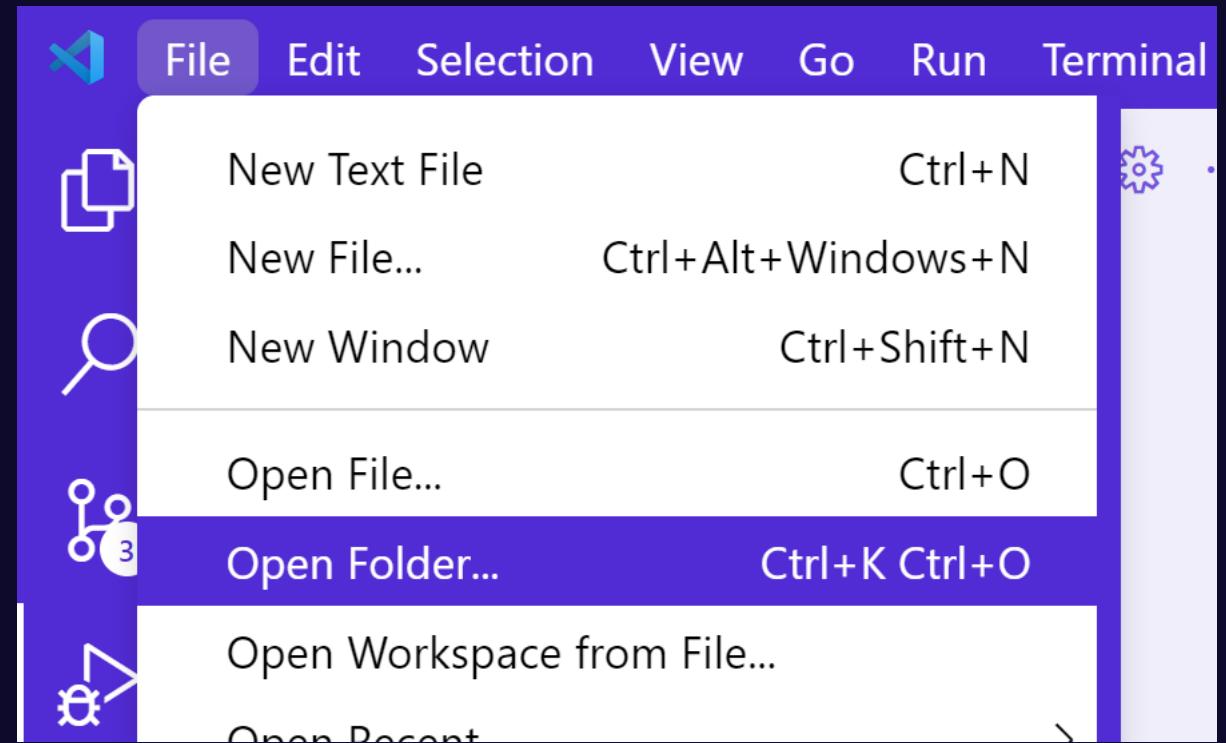
Now lets send it some telemetry from our DHT 11 sensor and Raspberry Pi

The screenshot shows the Azure IoT Central interface for a device named 'Samsung Lab Monitor'. The left sidebar has a 'Devices' section selected. The main area displays the device's details: 'Last data received: N/A', 'Status: Registered', and 'Organization: IoT Hack Session 1'. Below this, there are tabs for 'About', 'Overview' (which is selected), 'Raw data', 'Mapped aliases', and 'Files'. The 'Overview' tab contains two sections: 'Temperature, Humidity' and 'Temperature' and 'Humidity'. Both sections show a message: 'No data found. Check your device or network connection, and make sure you're part of the device's org.'

This Requires an Azure Subscription!

Try it Out!

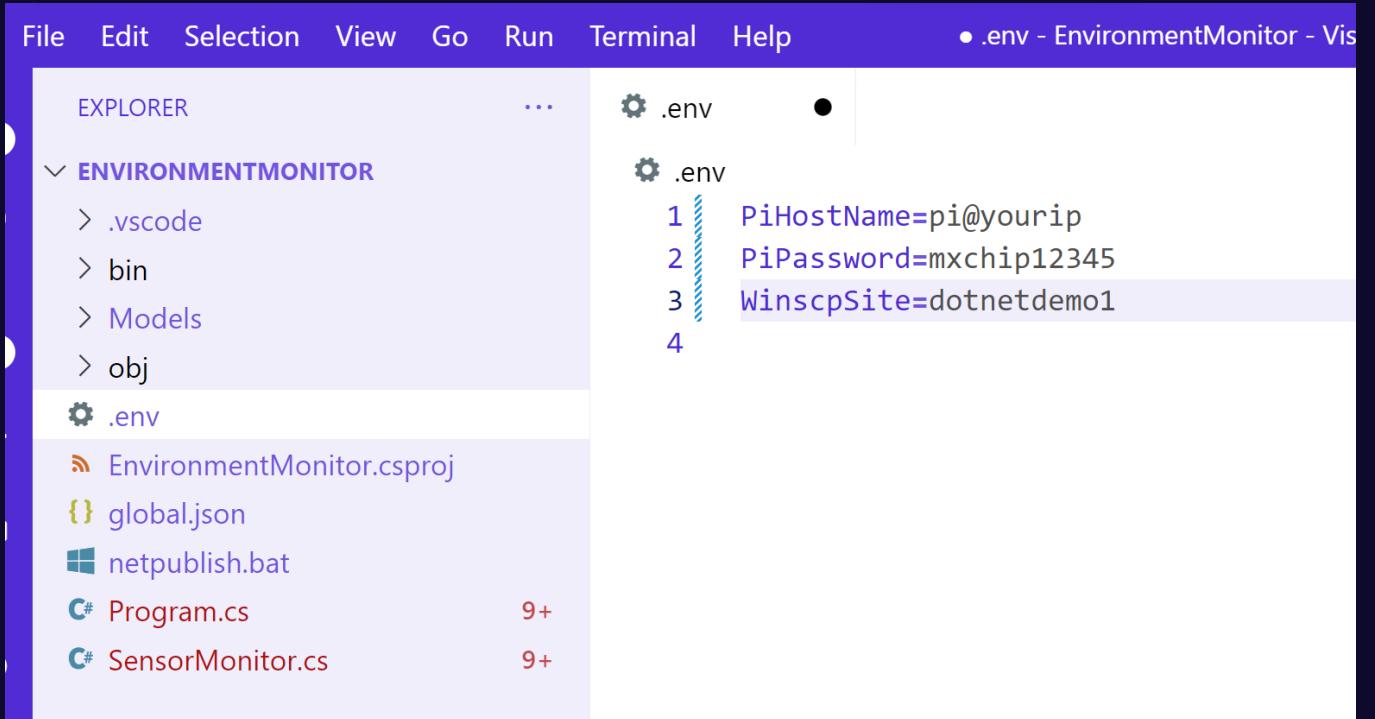
Open the Environment Monitor
folder in Visual Studio Code



This Requires an Azure Subscription!

Try it Out!

Configure the .env file
with your Raspberry Pi's
details



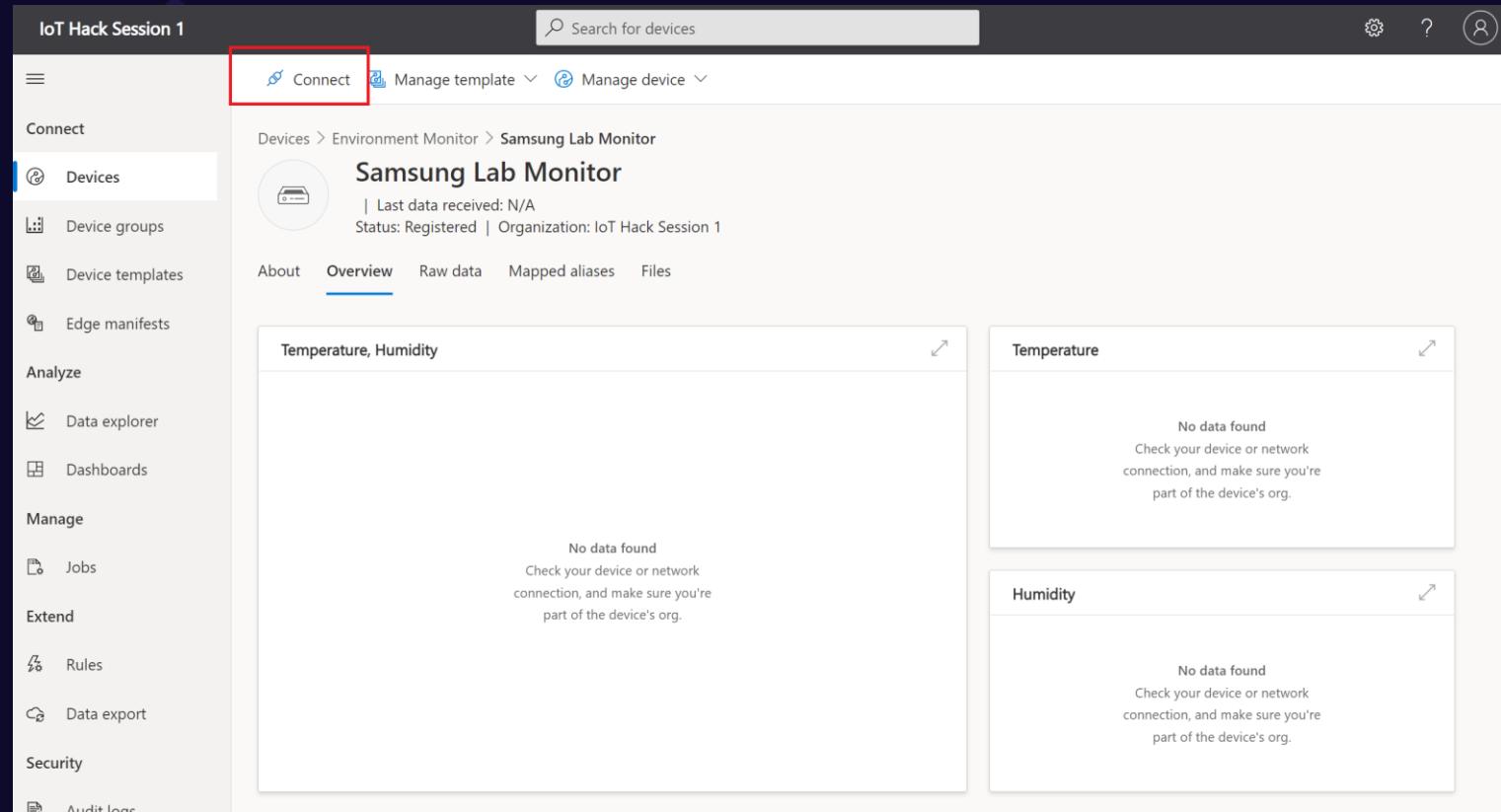
The screenshot shows the Visual Studio Code interface with the Explorer sidebar open. The sidebar lists project files: .vscode, bin, Models, obj, .env (selected), EnvironmentMonitor.csproj, global.json, netpublish.bat, Program.cs, and SensorMonitor.cs. In the main editor area, the .env file is displayed with the following content:

```
PiHostName=pi@yourip
PiPassword=mxchip12345
WinSCPsite=dotnetdemo1
```

This Requires an Azure Subscription!

Try it Out!

Within IoT Central select the device and press the “Connect” button.



This Requires an Azure Subscription!

Try it Out!

Take note of the:

ID Scope

Device ID

Primary Key

Device connection groups

ID scope ⓘ

One008: 

Device ID ⓘ

v42xge3m6q 

Choose the connection type for this device. You can change this later if you need to.

Authentication type

Shared access signature (SAS)

Key QR code

Shared Access Signatures (SAS) use security tokens and keys to connect to IoT Central. Use the SAS keys from the default enrollment group shown below to register your device. [Learn more](#) 

Primary key ⓘ

QWxDBujuvzSEbU+cI1jldSGp 

Secondary key ⓘ

re94U/LdGBZKAklOpopmxFkRBTW 

This Requires an Azure Subscription!

Try it Out!

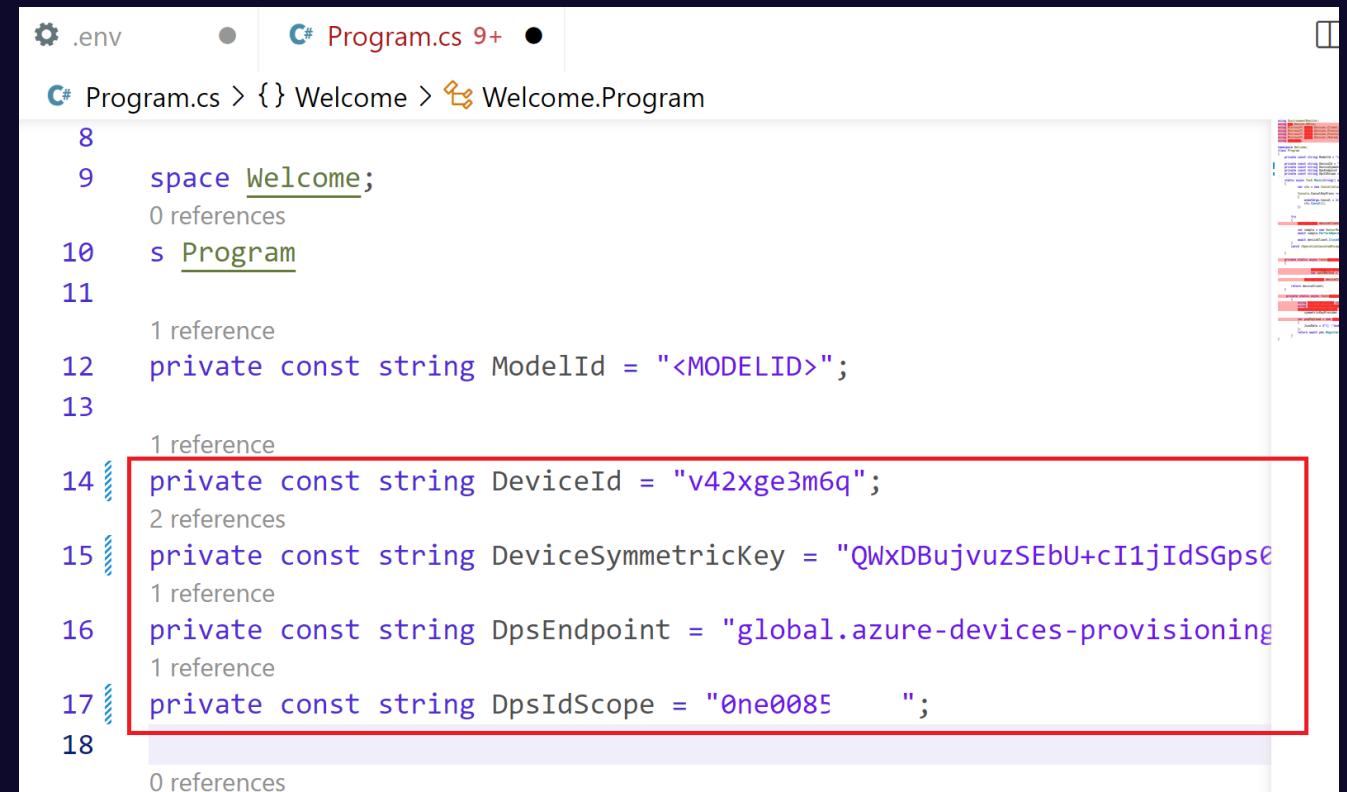
In Visual Studio Code fill in the
Program.cs file

ID Scope -> DpsIdScope

Device ID -> DeviceId

Primary Key ->

DeviceSymmetricKey



```
8
9     space Welcome;
0 references
10    s Program
11
12    1 reference
13    private const string ModelId = "<MODELID>";
14    1 reference
15    private const string DeviceId = "v42xge3m6q";
2 references
16    private const string DeviceSymmetricKey = "QWxDBuJvuzSEbU+cI1jIdSGpsE
1 reference
17    private const string DpsEndpoint = "global.azure-devices-provisioning
1 reference
18    private const string DpsIdScope = "0ne0085      ";
```

This Requires an Azure Subscription!

Try it Out!

Within IoT Central take note of
the

Interface @id

The screenshot shows the Azure IoT Central interface. On the left, a sidebar menu includes options like Connect, Devices, Device groups (highlighted with a red box), Device templates (also highlighted with a red box), Edge manifests, Analyze, Data explorer, Dashboards, Manage, Jobs, and Extend. The main content area displays a device template named "Environment Monitor". The top navigation bar shows "Device templates > Environment Monitor > Model > Environment Monitor". Below the title, it says "Application updated: 17 minutes ago" and "Interfaces published: 17 minutes ago". The "Edit identity" button is highlighted with a red box. The "Identity" panel on the right contains fields for Display name (Environment Monitor), Namespace (iotHackSession1), Name (EnvironmentMonitor_12r), Version (1), and Interface @id (dttm:iotHackSession1:EnvironmentMonitor_12r). A blue "Save" button is visible at the bottom of the Identity panel.

This Requires an Azure Subscription!

Try it Out!

In Visual Studio Code fill in the Programs.cs file

Interface @id -> ModelId

```
8
9  namespace Welcome;
0 references
10 class Program
11 {
12     private const string ModelId = "dtmi:iotHackSession1:Environmental";
13
14     private const string DeviceId = "v42xge3m6q";
15     private const string DeviceSymmetricKey = "QWxDBuJvuzSEbU+cI1jId9";
16     private const string DpsEndpoint = "global.azure-devices-provisioning";
17
18 }
```

This Requires an Azure Subscription!

Try it Out!

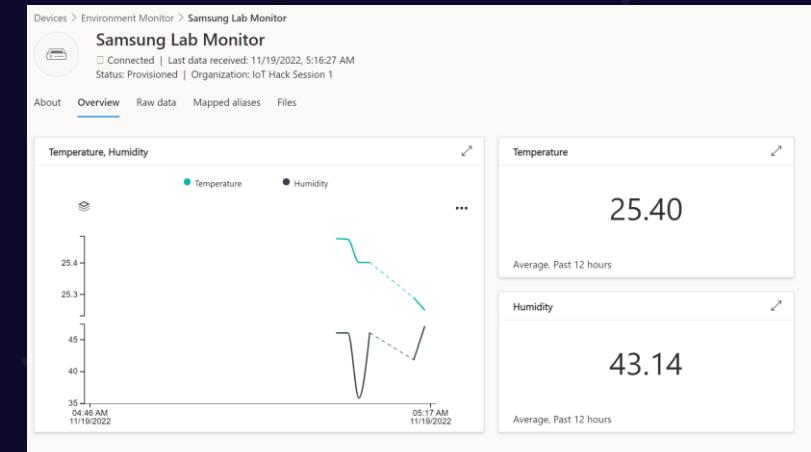
In Visual Studio Code fill in the
Program.cs file

Interface @id -> ModelId

The screenshot shows the Visual Studio Code interface with the following details:

- RUN AND DEBUG tab:** .NET Remote Launch - Raspberry Pi (highlighted with a red box).
- Code Editor:** Program.cs (highlighted with a red box). The code contains:

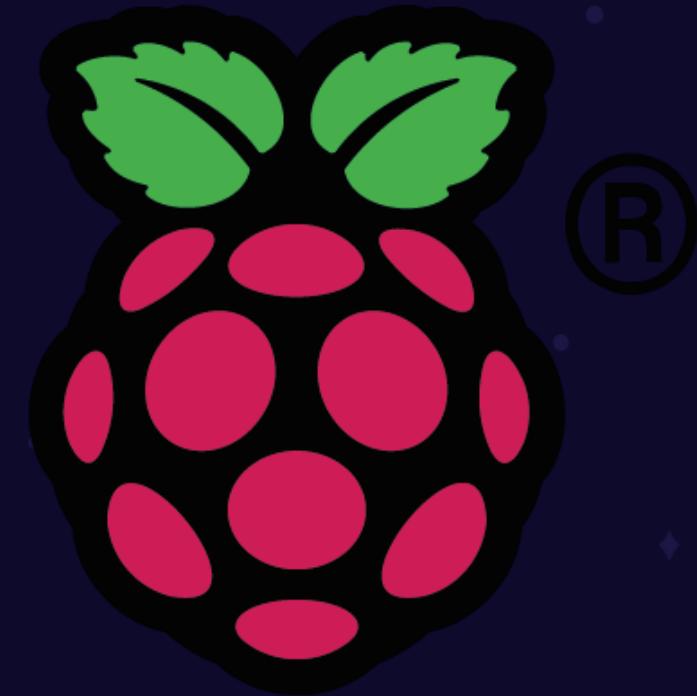
```
8
9  namespace Welcome;
0 references
10 class Program
11 {
12     private const string ModelId = "dtmi:iotHackSession;1";
13     1 reference
14     private const string DeviceId = "v42xge3m6q";
2 references
15     private const string DeviceSymmetricKey = "QWxL
1 reference
16     private const string DpsEndpoint = "global.azure
1 reference
17     private const string DpsIdScope = "0ne008567AB"
18 }
```
- OUTPUT panel:** Telemetry: Sent - {"Temperature":25.4,"Humidity":46} Humidity: 46% Temperature: 25,4C Telemetry: Sent - {"Temperature":25.4,"Humidity":46} Humidity: 46% Temperature: 25,4C Telemetry: Sent - {"Temperature":25.4,"Humidity":46}



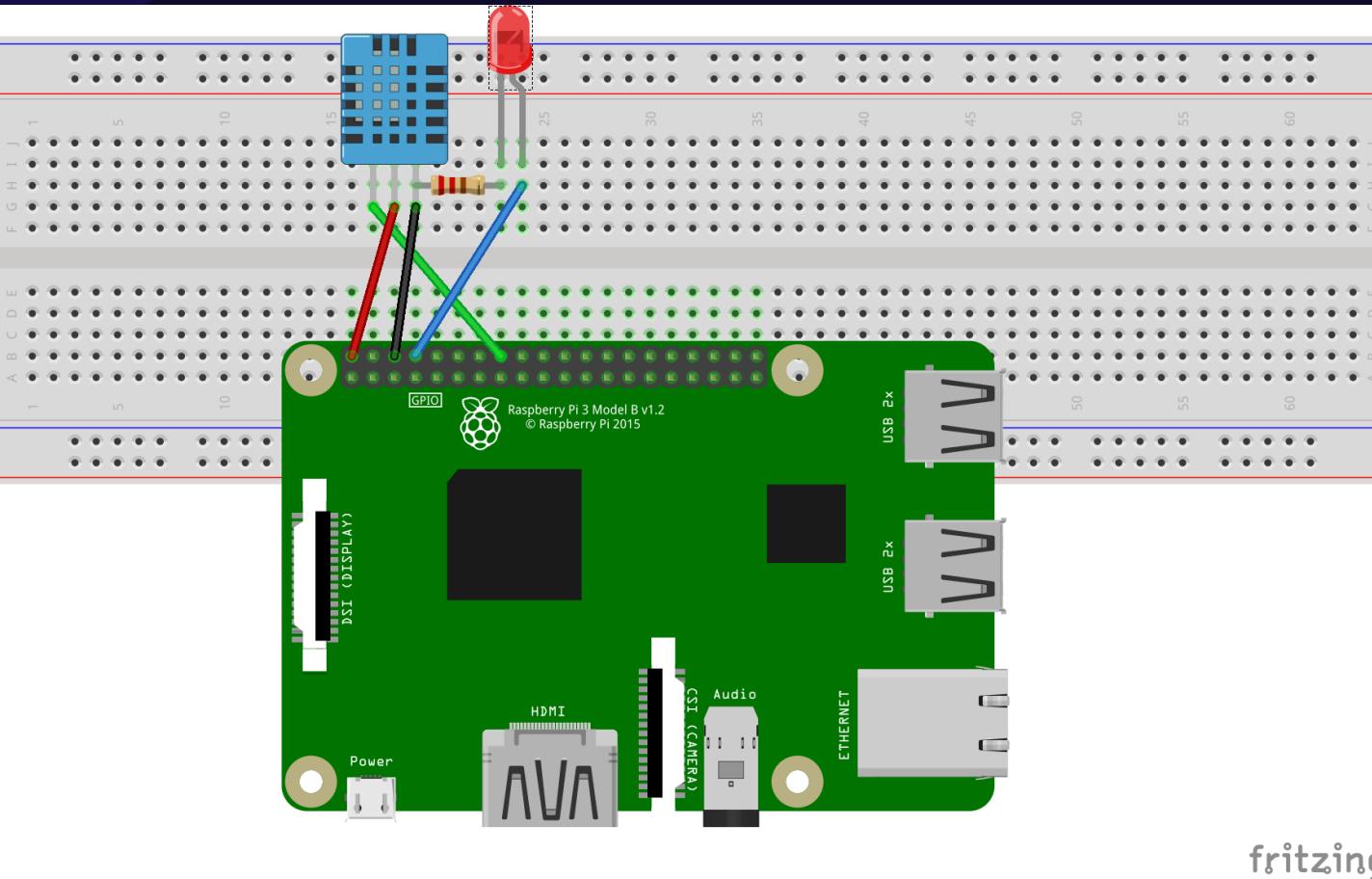
This Requires an Azure Subscription!

Module 4

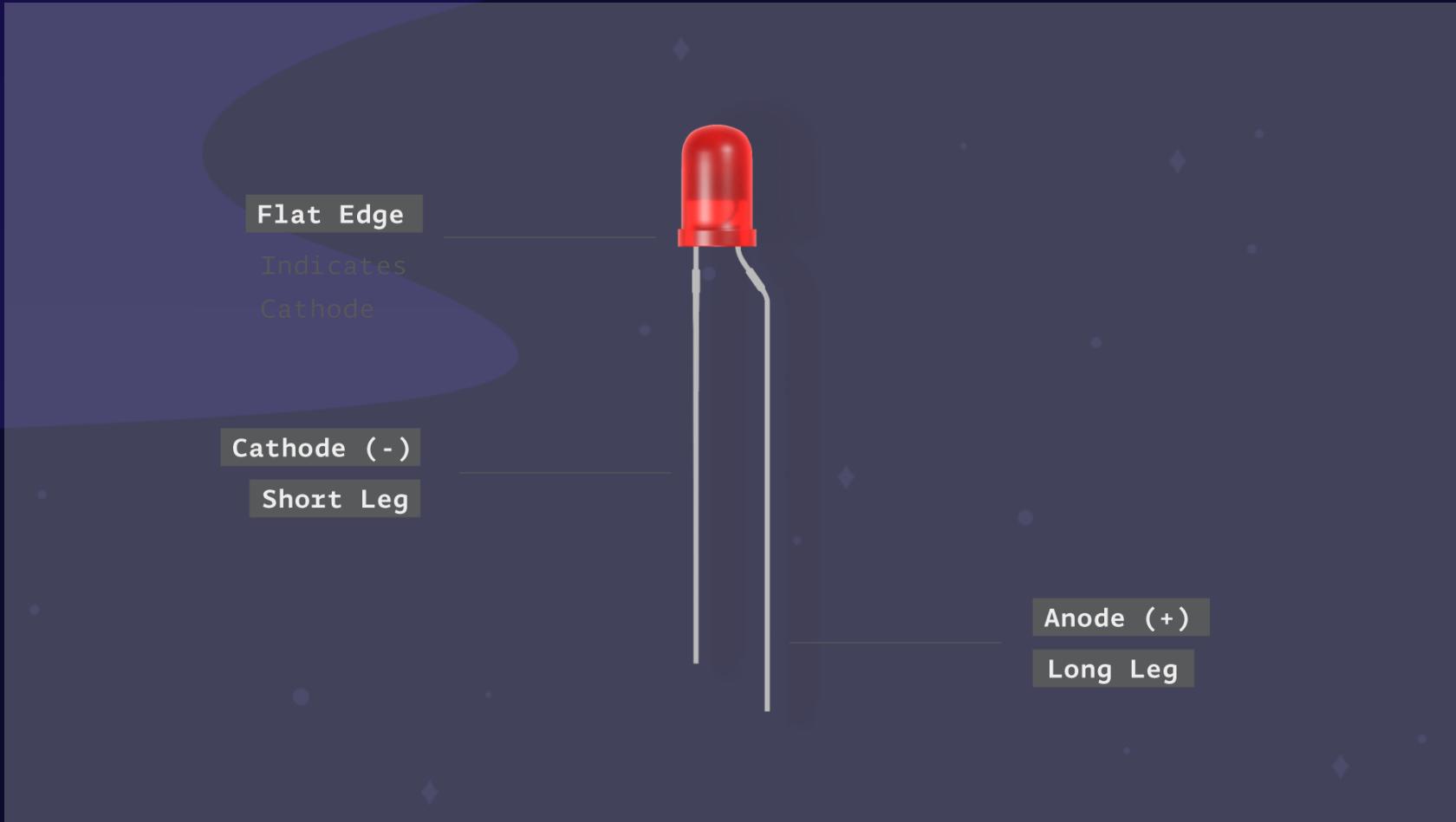
LED Fun



IoT is not IoT without a LED!



IoT is not IoT without a LED!



IoT is not IoT without a LED!

Why the resistor?

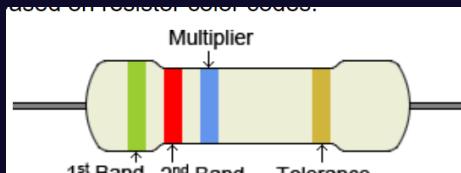
In this circuit the resistor is used to reduce current flow.

The Raspberry Pi is designed to only provide 23 mA max current. A resistor is important to not damage both the Raspberry Pi and the LED

<https://www.calculator.net/resistor-calculator.html>

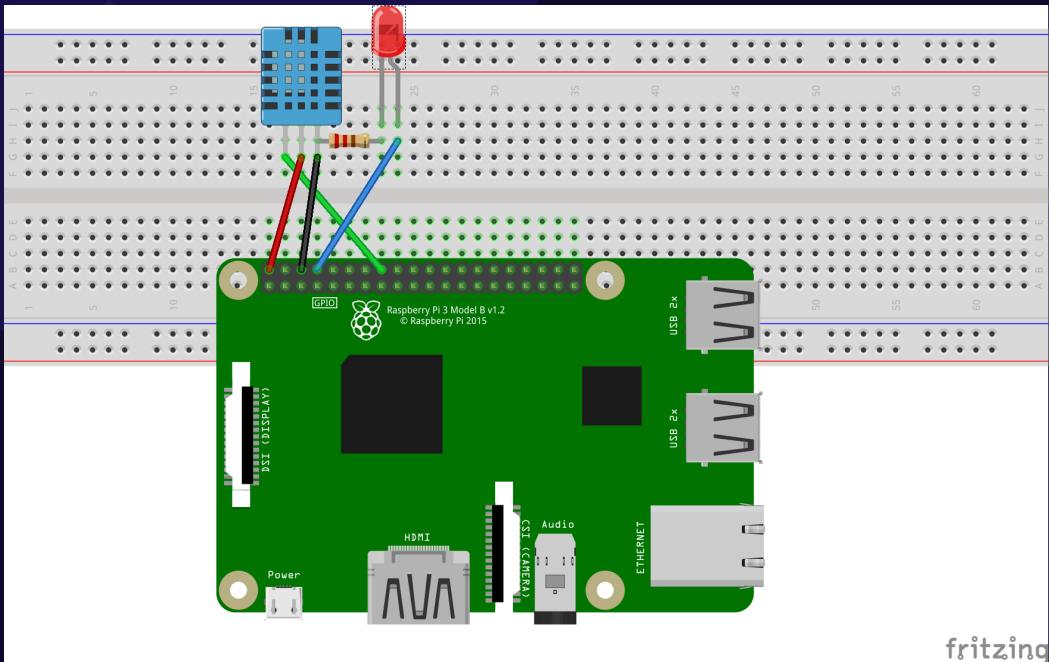
$$(V_{\text{supply}} - V_F) / I_F = (3.3 - 1.7) / 7\text{mA} = 220 \Omega$$

Based on resistor color codes.



Color	1 st , 2 nd Band Significant Figures	Multiplier	Tolerance
Black	0	$\times 1$	
Brown	1	$\times 10$	$\pm 1\% (\text{F})$
Red	2	$\times 100$	$\pm 2\% (\text{G})$
Orange	3	$\times 1\text{K}$	$\pm 0.05\% (\text{W})$
Yellow	4	$\times 10\text{K}$	$\pm 0.02\% (\text{P})$
Green	5	$\times 100\text{K}$	$\pm 0.5\% (\text{D})$
Blue	6	$\times 1\text{M}$	$\pm 0.25\% (\text{C})$
Violet	7	$\times 10\text{M}$	$\pm 0.1\% (\text{B})$
Grey	8	$\times 100\text{M}$	$\pm 0.01\% (\text{L})$
White	9	$\times 1\text{G}$	
Gold		$\times 0.1$	$\pm 5\% (\text{J})$
Silver		$\times 0.01$	$\pm 10\% (\text{K})$

IoT is not IoT without a LED!



Using the component pack, build the circuit using your Raspberry Pi, breadboard, LED, resistor and wires.

Connect resistor from ground (on DHT11) to LED cathode (short leg)

GPIO Pin 14 from the Raspberry Pi to LED anode

Try it out!

Coding Challenge

Turn the LED on when a certain temperature is reached. (check ambient temperature and use that as a starting point)

Hints

```
int ledPin = 14;  
  
var controller = new GpioController();  
  
controller.OpenPin(ledPin,PinMode.Output)  
  
controller.Write(ledPin,PinValue.High);
```

<https://learn.microsoft.com/en-us/dotnet/api/system.device.gpio.gpiocontroller?view=iot-dotnet-latest>

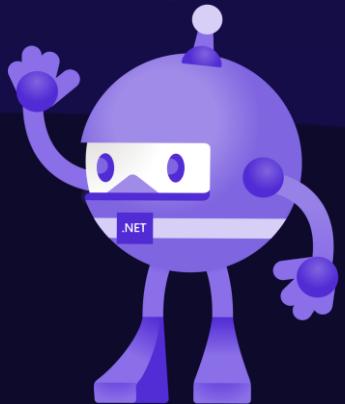
System.Device.Gpio



Resources

- Twitter: [@adpead](https://twitter.com/adpead)
- About.me: https://about.me/allan_pead
- Linkedin: <https://www.linkedin.com/in/adpead/>
- Blog: <https://explorationspace.co.za>
- Raspberry Pi South Africa
- <https://www.facebook.com/groups/1493503984198019>

Thank you!!



Getting Started with .NET

<https://learn.microsoft.com/en-us/dotnet/standard/get-started>

Visual Studio Remote Debugger

<https://www.hanselman.com/blog/remote-debugging-with-vs-code-on-windows-to-a-raspberry-pi-using-net-core-on-arm>

Getting Started with .NET Nanoframework

<https://docs.nanoframework.net/content/getting-started-guides/getting-started-managed.html>

