Extracted from:

Automate Your Home Using Go

Build a Personal Data Center with Raspberry Pi, Docker, Prometheus, and Grafana

The Pragmatic Bookshelf

Dallas, Texas

The Pragmatic Programmers

Automate Your Home Using Go

Build a Personal Data Center with Raspberry Pi, Docker, Prometheus, and Grafana

Ricardo Gerardi and Mike Riley

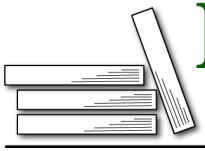
edited by Jacquelyn Carter

Automate Your Home Using Go

Build a Personal Data Center with Raspberry Pi, Docker, Prometheus, and Grafana

Ricardo Gerardi
Mike Riley

The Pragmatic Bookshelf
Dallas, Texas



Pragmatic Bookshelf

For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit *https://pragprog.com*.

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 979-8-88865-050-9
Encoded using the finest acid-free high-entropy binary digits.
Book version: B1.0—November 1, 2023

Now that you have our IT Infrastructure in a Box configured and ready to accept inbound data, you can begin building our first home automation project: a networked temperature monitor. This project will use the Raspberry Pi Pico W's on-board temperature sensor to report the current ambient temperature around the sensor. The Pico W will then communicate with the Raspberry Pi that is running the Prometheus server that you set up previously to poll a web server running on the Pico W. This web server will provide these temperature values in both Celsius and Fahrenheit measurement metrics.

Once the sensor is calibrated and reporting its results via a REST-accessible JSON payload, you can deploy the Pico W in a number of environments such as a basement, a freezer, or even outdoors in order to obtain a reoccurring series of temperature updates. The Raspberry Pi you set up with Prometheus and Grafana in the previous chapter will reach out and poll the Pico W at set intervals. The formatted JSON data obtained from the Pico W will then be consumed by our Prometheus instance and visualized by our Grafana instance on the Raspberry Pi server. By visualizing the data, it will be easy to spot trends and changes in temperature, as well as assign alerts when defined thresholds are exceeded. For example, you can configure Grafana to email you when your freezer temperature goes higher than 0 degrees Celsius (32 degree Fahrenheit). That alert could save you from spoiling a lot of frozen food!

Remarkably, there is also work currently underway to make an minimized version of the Go language, called TinyGo[1] that will be capable of running on the Pi Pico W as well. Unfortunately at the time of writing this book, TinyGo was incompatible with the Pico W. The TinyGo project also had not yet ported Go's HTTP libraries to TinyGo, making web-based REST calls to a Pico W running TinyGo a significant, unnecessary burden. As such, we decided to use MicroPython for our Pi Pico W web interactions. If and when TinyGo eventually matches parity with MicroPython's functional HTTP and JSON libraries running on a Pico W, we may revisit using TinyGo instead of MicroPython in a future edition. But for now, MicroPython is the most efficient and easiest Pi Pico W microcontroller language for the job. To learn more about MicroPython and its Read-Evaluate-Print-Loop (REPL), refer to the Raspberry Pi Pico Python SDK[2] documentation.

Now it's time to roll up your sleeves, gather up the necessary hardware, and start building the solution!

1. https://tinygo.org/
2. https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-python-sdk.pdf

Creating A Pico W Development Environment

The Raspberry Pi Pico W is a WiFi-enabled version of their popular Pico micro-controller design. Unlike the Raspberry Pi which can be reached via a variety of network protocol connections, such as SSH, the Pico W is a micro-controller, and thus does not have the storage, memory, or operating system capacity of a Linux distribution. Instead, it is designed to immediately start a runtime at power on, and execute whatever program and script it is instructed to do so. The Pico W currently supports its own C++ libraries and a minimal variant of Python called MicroPython.[3]

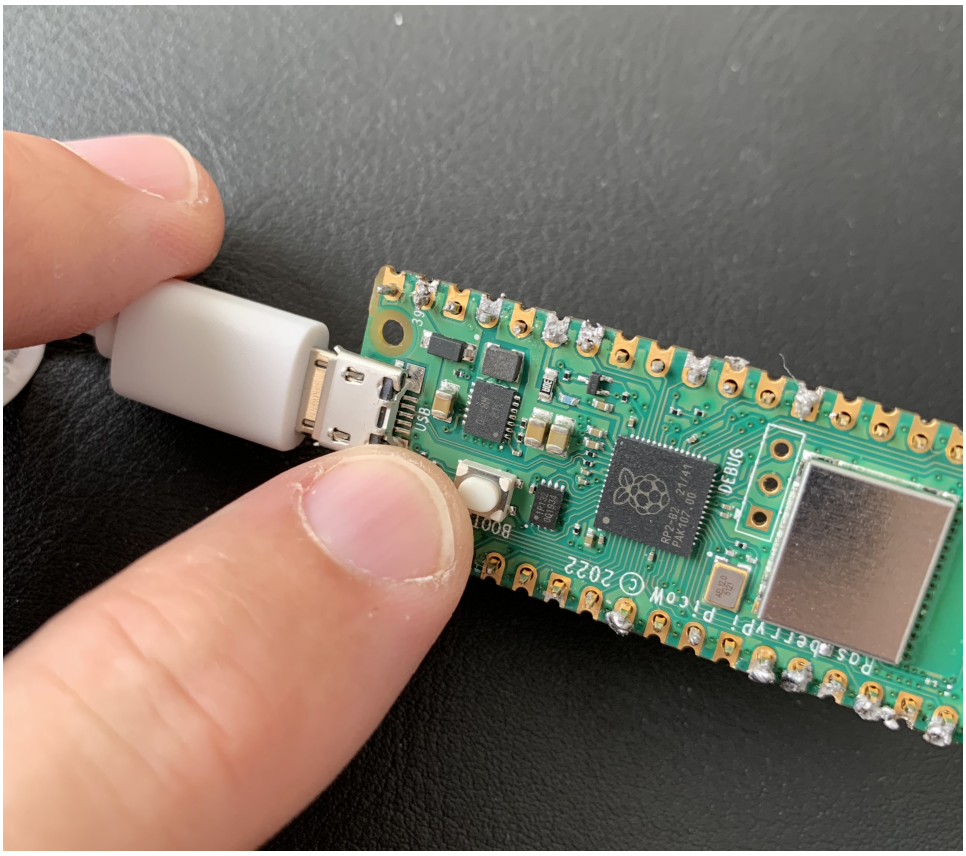
We will take advantage of this wireless network-enabled Pico W version so that the Pico W can be placed anywhere there is a WiFi signal it can associate with, and a power supply it can connect to. However, because the Pico W is a micro-controller and not a full-blown Linux-based server, we have to do some additional work to connect to and program for it.

The ideal development environment for a Pico W board is a desktop PC or laptop running Linux, macOS, or Windows, and a USB to USB micro cable to connect the Pico W to the computer.

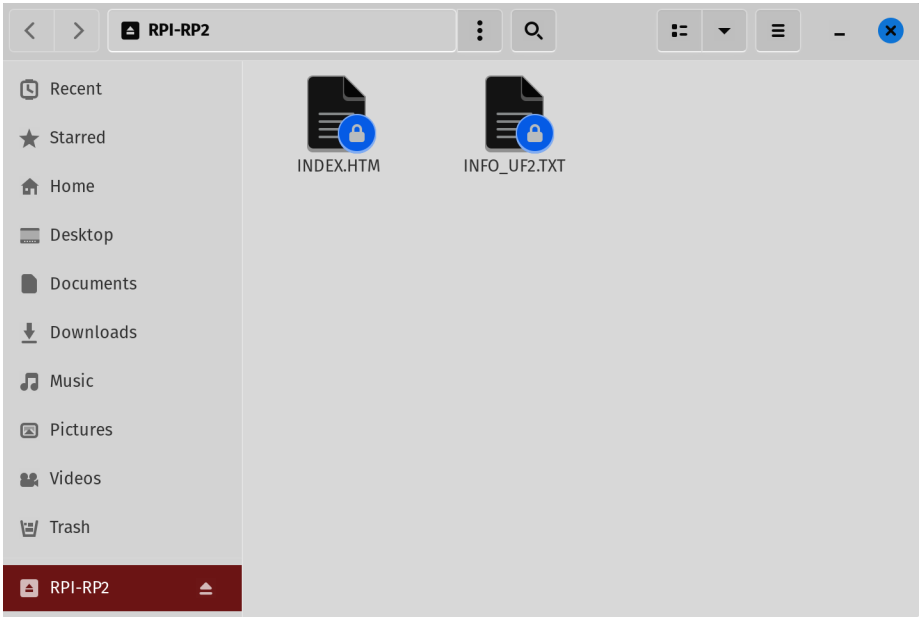
We're going to install Thonny,[4] a particularly useful Python IDE that also easily communicates with the Pi Pico and Pico W hardware. Thonny runs on Linux, macOS, and Windows, so downloading and installing this Python-centric IDE on these platforms should work identically. In addition to being a useful Python editor, Thonny knows how to connect to and communicate with the Pico W board, making it a one-stop IDE for Pico W program development.

Before you can load and execute MicroPython scripts on the Pico W, you must first install the MicroPython runtime. Following the installation instructions on the Raspberry Pi's Pico documentation website,[5] download the UF2 file that matches your Pico model (the original Pico uses a different UF2 file compared to the Pico W). Once the correct UF2 file has been downloaded, hold down the white BOOTSEL button on the Pi Pico W while plugging in a USB cable between the Pico W and your PC, as shown in the next photo.

3. https://www.raspberrypi.com/documentation/microcontrollers/micropython.html
4. https://thonny.org/
5. https://www.raspberrypi.com/documentation/microcontrollers/micropython.html

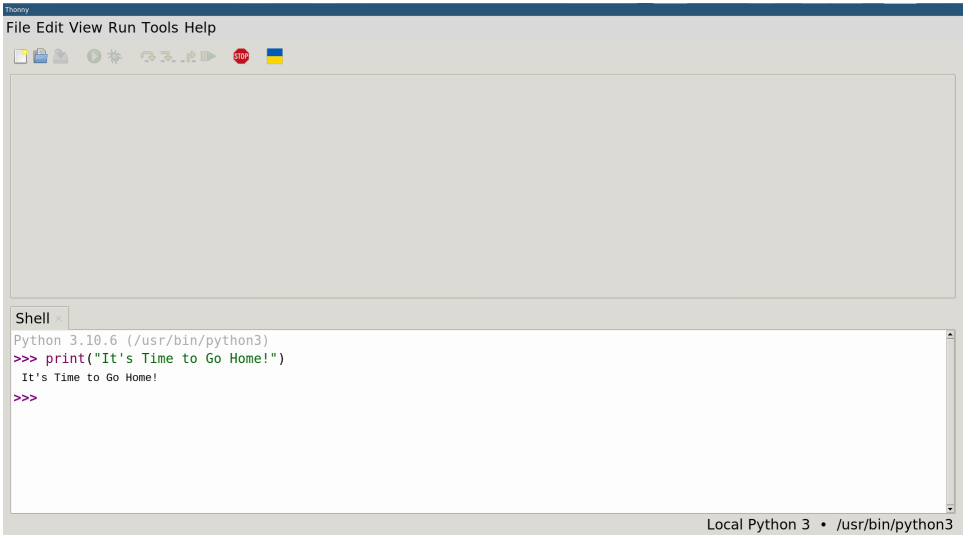


If successful, the PC should see the Pico W as a mountable USB drive, as indicated in the following screen capture.



Drag the freshly downloaded MicroPython UF2 file into the mounted Pico W's USB drive. The Pico W will automatically recognize this as a special file type by installing the file and rebooting the Pico W (and thereby ejecting its mounted USB drive in the process).

With the MicroPython runtime installed, launch the Thonny IDE and verify that "MicroPython (Raspberry Pi Pico)" is detected in the lower right corner of the Thonny IDE window, as shown in the following screenshot.



If you don't see that, make sure your Pi Pico W is attached to your PC with the USB cable, and then select that lower right corner section of the Thonny IDE. Doing so will pop up a list of the various Python runtimes Thonny has identified on your computer. One of those selections should be the MicroPython runtime that was installed on the Pi Pico W.

Once properly identified and selected, press the red button located in Thonny's toolbar to reset any running Python processes as well as verify communications with the Pi Pico W board. To verify that everything is working correctly, enter a simple `print()` command in the Shell window, such as:

```
print("It's Time to Go Home!")
```

Select the Return/Enter key on your keyboard, and if Thonny is properly connected to and communicating with the Pi Pico W board, you should see "It's Time to Go Home!" reprinted below the line of code you entered into Thonny's shell window.

Now that you have a working MicroPyhon runtime installed and executing on the Pi Pico W, you can begin writing a REST server that will poll the Pico W's on-board temperature and report that value in both Celsius and Fahrenheit, formatted in a JSON payload that can be consumed for further analysis. The values in this JSON will be converted into Prometheus-friendly formatting using a Go program that we will write to perform the polling and conversion. But first, we need to get the temperature value off the Pico W's on-board temperature sensor, format it into JSON-friendly format, and have a simple HTTP server ready to accept new connections and deliver the JSON payload.