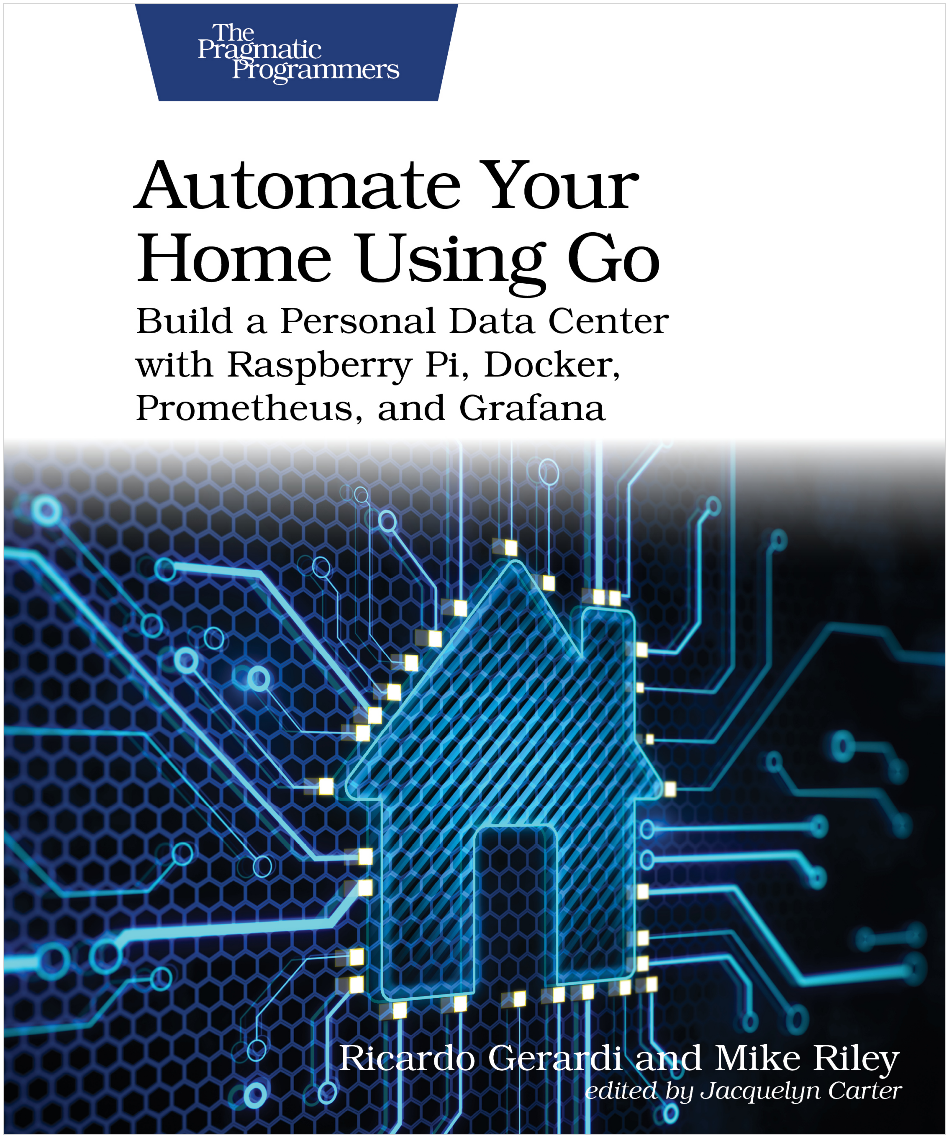


The Pragmatic Programmers

# 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*

Now that you have your IT Infrastructure in a Box configured and ready to accept inbound data, you can begin building your first home automation project: a networked temperature monitor. This project will use the Raspberry Pi Pico W's onboard 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.

## Project's Hardware Requirements

This project requires these components:

- *Raspberry Pi Pico W:* The microcontroller with onboard Wi-Fi and a temperature sensor to report ambient temperature.
- *Raspberry Pi server:* A Raspberry Pi 3, 4, or Zero 2 to run the Prometheus exporter to scrape data from the Pico W.

For more details consult Selecting a Raspberry Pi, on page ?.

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 your Prometheus instance and visualized by your Grafana instance on the Raspberry Pi server. By visualizing the data, it'll 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 degrees Fahrenheit). That alert could save you from spoiling a lot of frozen food!

Remarkably, there's also work currently underway to make a minimized version of the Go language, called TinyGo[1] that's capable of running on the Pi Pico W as well. At the time of writing this book, the Wi-Fi driver for the Pico W is not officially available with TinyGo, but the development version cyw43439[2] is available. We'll use this driver for the book's project. When the driver is officially integrated within TinyGo, the code will likely not change dramatically, but the driver´s import path might change.

1. https://tinygo.org/
2. https://github.com/soypat/cyw43439

By the end of this chapter, you'll understand how to work with the Pico W device, and how to use TinyGo to develop Go programs that fit many microcontrollers. These important skills will allow you to develop other projects that use microcontrollers to handle hardware, connectivity, and logic in places or circumstances where it's harder to use larger, more power hungry, devices.

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

## Understanding the Pico W Device

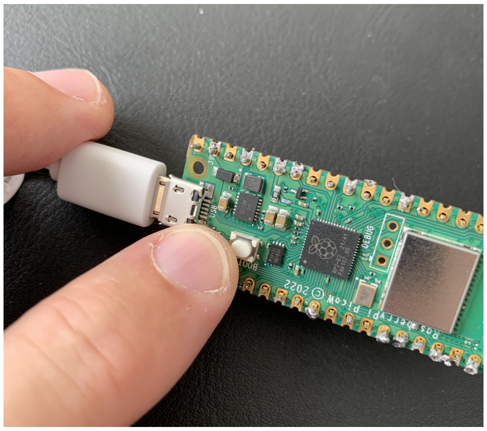
The Raspberry Pi Pico W is a Wi-Fi-enabled version of their popular Pico microcontroller design. Unlike the Raspberry Pi which runs a full Linux operating system and can be reached via a variety of network protocol connections, such as SSH, the Pico W is a microcontroller, and thus does not have the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

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

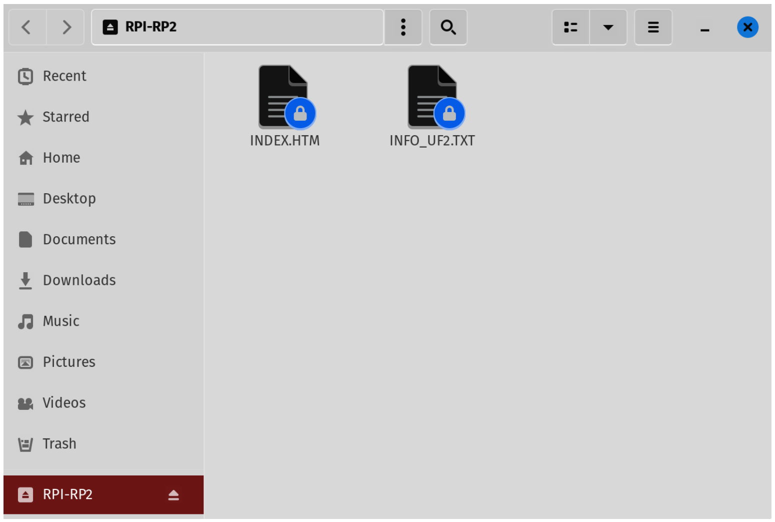
The ideal development environment for a Pico W board is a desktop PC or laptop running Linux, macOS, or Windows, and a USB to USB micro cable to connect the Pico W to the computer. You can develop TinyGo programs using your preferred IDE or text editor. When you're done, you'll use the `tinygo` command-line application to compile your program into a UF2 firmware image compatible with the Pico W. Install TinyGo on your development machine by following the instructions on the official Quick Install Guide.[4]

To transfer this image to the Pico W device, you need to start the Pico W in file transfer mode by holding 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://tinygo.org/
4. https://tinygo.org/getting-started/install/



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



Drag the freshly compiled 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).

If everything goes well, your program will execute on the Pico W automatically. You can use TinyGo's monitor subcommand to monitor the Pico W serial interface for logs to ensure the program is running.

Now that you understand how to develop and transfer Go programs to the Pi Pico W, you can begin writing a REST server that will poll the Pico W's onboard 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'll write to perform the polling and conversion. But first, we need to get the temperature value off the Pico W's onboard temperature sensor, format it into JSON-friendly format, and have a simple HTTP server ready to accept new connections and deliver the JSON payload.