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

This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

In Chapter 4, Networking a Temperature Monitor, on page ?, we created a network-enabled temperature probe to report current temperature conditions. While we could use that same approach for monitoring outdoor temperatures as well, the Pico W would need to be shielded in a weather-resistant case. If the external temperatures became extreme, the Pico W might permanently fail. Fortunately, enough external temperature sensors are already monitoring outdoor weather conditions. We can simply poll available APIs for JSON payloads containing current weather condition values. But rather than simply report the number, we can use colored bulb lighting to visually indicate whether it's cold, comfortable, or hot outside. We'll send color commands to a Philips Hue lighting setup based on the temperature values received by the outdoor temperature API call.

Project's Hardware Requirements

This project requires these components:

- *Raspberry Pi server:* A Raspberry Pi 3, 4, or Zero 2 to act as the application server.
- *Hue base station:* Part of the Hue Start Kit, it maintains the inventory and state of the Hue lighting in your home.
- *Hue multi-colored lighting strip:* The light we'll program based on the outdoor temperature.

For more details, consult Adding Other Hardware Components, on page ?.

After you complete this project, it's going to look like the picture on page 4, where you can see the light with two different states, blue and red.

If the light is blue, grab a jacket because it's cold outside. Is the light red? Then it's warm enough to enjoy the outdoors wearing a T-shirt.

In this chapter, you'll use REST API calls to obtain the external weather information and control the lights. Handling APIs is an important skill to learn as it allows you to obtain information from a large pool of sources. It also allows you to control automation devices that expose such API interfaces, expanding the range of home automation controllers you can use on your own projects.

Let's get started.

Polling the Weather

To query the current weather conditions, we need access to an API that can provide those details. Fortunately, a service called OpenWeather[1] offers a free tier for developers that allows a copious number of calls to their service. Sign up[2] to request a free API key. You'll need this key when making calls to OpenWeather's API. In particular, it'll be used to poll the current outdoor temperature in your area.

Once you have your free OpenWeather API key, test it out by polling the current temperature in your area with this small Go program, which uses the `openweathermap` package to query the OpenWeather API. Replace the API_KEY and ZIP_CODE values with your own before running the test. If you live outside the United States, replace the country code as well:

```
package main

import (
    "fmt"
    "log"
    "os"

    owm "github.com/briandowns/openweathermap"
```

1. https://openweathermap.org/
2. https://home.openweathermap.org/users/sign_up

```
)

func main() {
    w, err := owm.NewCurrent("F", "EN", API_KEY)
    if err != nil {
        log.Fatalln(err)
    }

    w.CurrentByZip(ZIP_CODE, "US")
    fmt.Println(w.Main.Temp)
}
```

Save the file as `openweathertest.go` and run `go mod tidy` to download GitHub user Brian Downs's openweathermap Go library. This library makes it very easy to use OpenWeather's API in the Go language environment.

With everything nice and tidy, run the program using the usual Go run syntax to test your API key, like this:

```
$ go run openweathertest.go
```

Assuming the values you replaced for your API_KEY and ZIP_CODE are valid, you should see the current temperature output in Fahrenheit. If you prefer the temperature scale to be reported in Celsius, change the parameter in `NewCurrent` to `C`, like this:

```
w, err := owm.NewCurrent("C", "EN", API_KEY)
```

Congratulations! You're now able to poll the current outdoor temperature in your area. The OpenWeather API offers many other options you can explore. The free tier is somewhat limited in the level of detail and forecast information it provides, but enough data is available to be useful for our project. Feel free to experiment with other calls to the API, as well as poll other geographic regions where you might be interested in the current temperature.

In the next section, we'll use the current temperature value received from the `w.Main.Temp` variable and light a Hue Philips color lightstrip to visually reflect the current outdoor temperature.

Changing the Color

The Hue base station maintains the inventory and state of the Hue lighting in your home, and the lighting strip is what we'll program based on the outdoor temperature. In order to make the color of the lighting meaningful, we need to determine temperature ranges with which to display the appropriate color. For example, the color blue is frequently associated with cold. So setting the light strip to that color anytime the temperature is below 50 degrees Fahrenheit would indicate cooler temperatures outside. Conversely, the color red

typically indicates hot. Thus, anytime the outdoor temperature is hotter than 90 degrees, change the light strip color to red. Here are the color recommendations between those two values:

```
Blue   = Below 50 degrees
Yellow = Between 51 and 65 degrees
Green  = Between 66 and 79 degrees
Orange = Between 80 and 89 degrees
Red    = Above 90 degrees
```

Let's codify those rules in Go using a switch statement and append it to the openweathertest.go program to test. To make it easier to code the switch statement, first assign the temperature returned by the API call w.Main.Temp to a new variable currentTemp in the main function:

```
var currentTemp = w.Main.Temp
```

Then, append this switch block to the end of the main function to display the color:

```
switch {
case currentTemp < 51:
    fmt.Println("Blue")
case currentTemp >= 51 && currentTemp < 66:
    fmt.Println("Yellow")
case currentTemp >= 66 && currentTemp < 80:
    fmt.Println("Green")
case currentTemp >= 80 && currentTemp < 90:
    fmt.Println("Orange")
case currentTemp >= 90:
    fmt.Println("Red")
}
```

Run the program via the usual go run openweathertest.go command, and depending on the current outdoor temperature, the appropriate color should display right after the actual temperature value that was evaluated. Now that the proper respective color is indicated based on the outside temperature, it's time to hook up and connect to the Hue base station.

Programming the Hue

Before we can start programming the Hue from a Go application, make sure that you have correctly set up the Hue base station on your network, and added the light strip to the Hue's inventory. You can use the official Philips Hue app, available from either the Android[3] or iOS[4] app stores.

3. https://play.google.com/store/apps/details?id=com.philips.lighting.hue2
4. https://apps.apple.com/ie/app/philips-hue/id1055281310

Once you can remotely control your Hue light strip from the Hue app, you are ready to configure a new user account on the Hue base station. You'll use this account to interact with and send commands from your Go application.

Several Hue libraries for Go are available on GitHub. The one that works best with this particular project was created by GitHub user Collinux, called gohue. This rudimentary library makes it easy to connect, control, and set basic colors on Hue lighting.

Before we can remotely control Hue-managed lights, we need an authorized User ID to log into the Hue base station. The gohue library provides a CreateUser function that instructs the Hue to generate a new User ID for this purpose. To do so, write the following Go program:

```
package main

import (
    "github.com/collinux/gohue"
)

func main() {
    bridgesOnNetwork, _ := hue.FindBridges()
    bridge := bridgesOnNetwork[0]
    username, _ := bridge.CreateUser("gohomeuser")
    fmt.Println(username)
}
```

Save the code as createhueuser.go and run it with the Hue base station nearby. You'll need to press the large button on the top of the Hue base station to authorize the User ID creation request when the createhueuser.go program is run.

```
$ go run createhueuser.go
```

Remember to copy the username ID that is generated after authorizing the request on the Hue. You'll use this ID for programmatic Hue base station access.

Now that you have created the new authorized Hue account, you can use it when programmatically manipulating your Hue lights. Verify that this newly generated User ID allows you to control your Hue light by creating a simple program that turns the light on. The following sample code assumes you named your light "Desk" using the Hue smartphone app. You can reference this light in your code using the function GetLightByName() and whatever actual name you assigned to the light using the Hue smartphone app.

```
package main

import (
    "github.com/collinux/gohue"
```

```
)

func main() {
    HUE_ID := os.Getenv("HUE_ID")
    HUE_IP_ADDRESS := os.Getenv("HUE_IP_ADDRESS")

    bridge, _ := hue.NewBridge(HUE_IP_ADDRESS)

    bridge.Login(HUE_ID)

    deskLight, _ := bridge.GetLightByName("Desk")

    deskLight.On()
}
```

Save the code as `huetest.go` and make sure your `HUE_ID` and `HUE_IP_ADDRESS` environment variables are properly assigned. Then run the code via the typical `go run` command.

```
$ go run huetest.go
```

If everything runs successfully, your targeted Hue light should turn on. Now that we can control lights from Go, let's expand our code to run as a service.