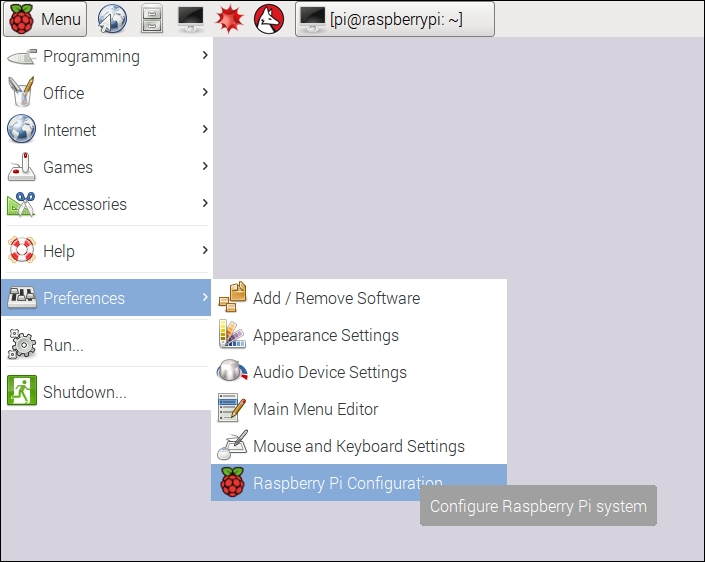
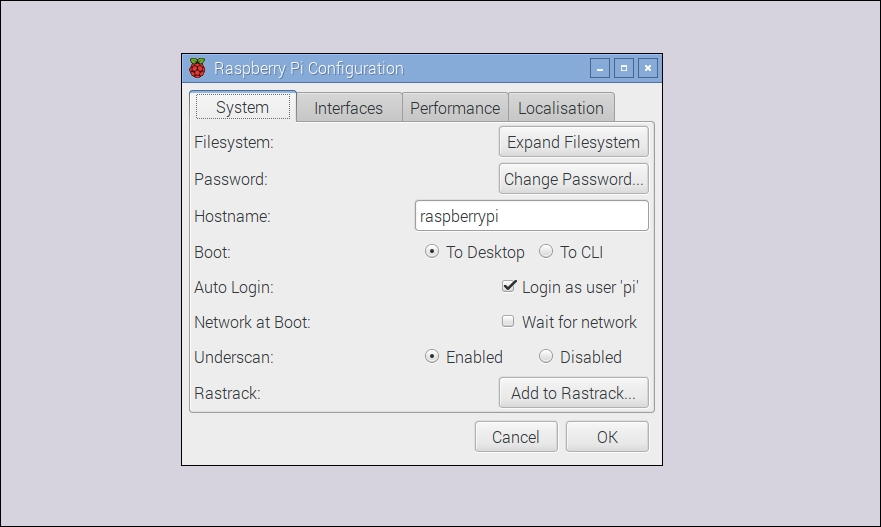
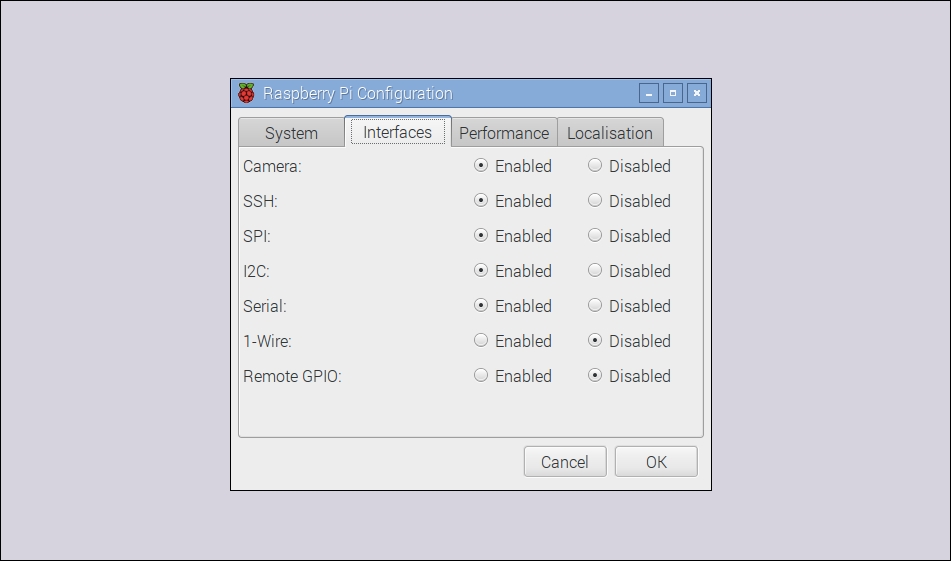
Next, we need to enter the Raspberry Pi configuration panel so we can set some essential settings. You can find the **Raspberry Pi Configuration** tool inside the main **Menu**:



Inside the configuration tool, first press on **Expand Filesystem**:



This will make sure that the Pi now has access to all the space available on the SD card. Also, click on the **Interfaces** tab and check that **SSL** is checked:



Inside the same configuration tool, you can also give a network name to your Pi. I simply called mine pizero.

We are now going to perform some tests from your computer to make sure that Raspberry Pi is correctly configured for remote access and that it has access to the Internet.

To do so, open a terminal window on your computer (or use PuTTY if you are using Windows), and type the following:

**ssh pi@pizero.local**

This should initiate a connection to your Pi board and ask for your password. Once you type in your password, you should now be connected to the Pi board. If that doesn't work at this point, try replacing the name of your Raspberry Pi with the IP address of the board (you can get this by typing ifconfig inside a Terminal on the Pi itself).

Then, from your computer, type the following:

**sudo apt-get update**

Then type the following command:

**sudo apt-get upgrade**

This will upgrade your Pi board by downloading all the latest packages from the official Raspberry Pi repository, so it's a great way to make sure that your board is connected to the Internet.

**Installing Node.js**

To finish this chapter, we are going to install Node.js, which is a powerful framework that we will use to run most of the applications that we are going to see inside this book. Luckily for us, installing Node.js on Raspberry Pi is really simple.

First, log into your Raspberry Pi via SSH. We are going to quickly check that Node.js was not installed with the Linux image. To do so, type the following:

**node -v**

If this returns the version of Node.js, you can stop there. If it returns an error, you will have to install it manually. To do so, first type the following:

**curl -sL https://deb.nodesource.com/setup\_4.x | sudo -E bash -**

After that, type the following command, which will install Node.js:

**sudo apt-get install -y nodejs**

Finally, install some additional tools with the following:

**sudo apt-get install -y build-essential**

You can now test that Node.js is correctly installed by typing the following:

**node -v**

Congratulations, you now have a fully configured Raspberry Pi Zero! In the next chapter, we are going to build your first application using the Zero board and learn how to measure data from sensors.

# Chapter 2. Measure Data Using Your Raspberry Pi Zero Board

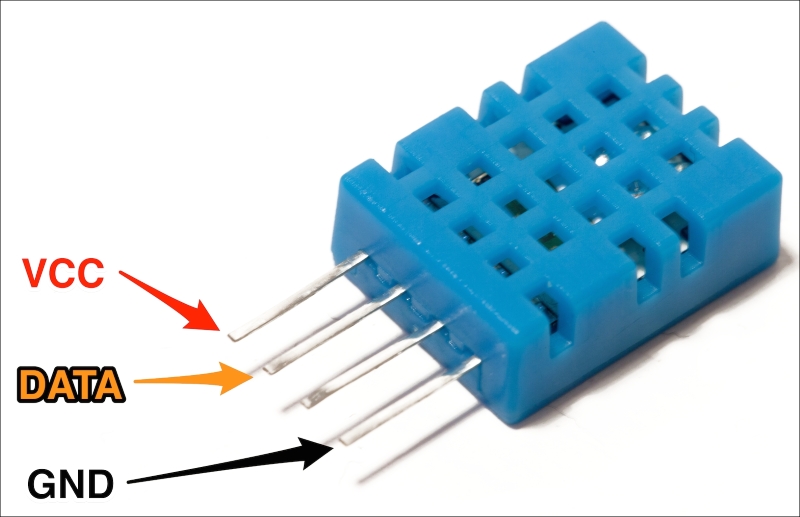
In the first chapter of this book, we worked on setting up your Raspberry Pi board so you can use it in your projects and realize all the projects you'll find in this book.

In this chapter, we are going to make our first project using the Zero board: measuring data using your board. We are going to learn how to connect a very simple temperature and humidity digital sensor to your Pi, and how to write software to read data from it.

From there, we'll look at some very basic applications using this sensor that can be really useful inside a smart home: how to log data on the Pi itself, how to access the measurements remotely, and finally, how to display past data on a nice plot.

# Hardware and software requirements

We have already discussed most of the requirements for this project in the first chapter of this book. Here, you will simply need an additional component: a DHT11 sensor (<https://www.adafruit.com/products/386>). The following image shows the sensor:



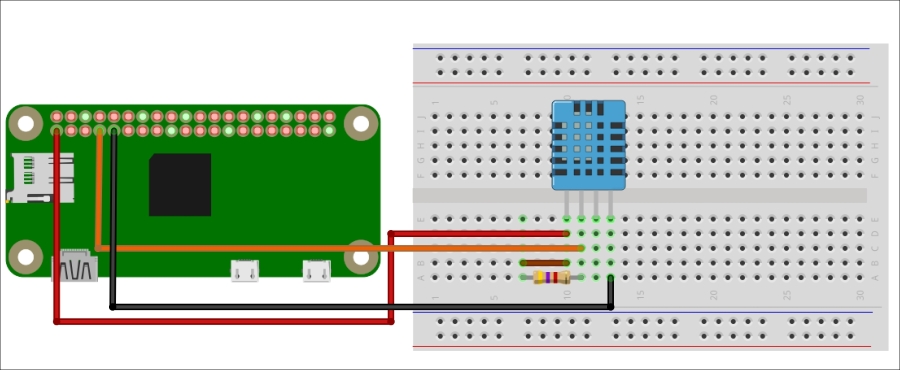
You can of course use other similar sensors, for example the DHT22, which is more precise. To use a DHT22, you will only need to change one thing inside the code we'll see later.

You will also need a 4.7k Ohm resistor to make the sensor work, as well as jumper wires and a breadboard.

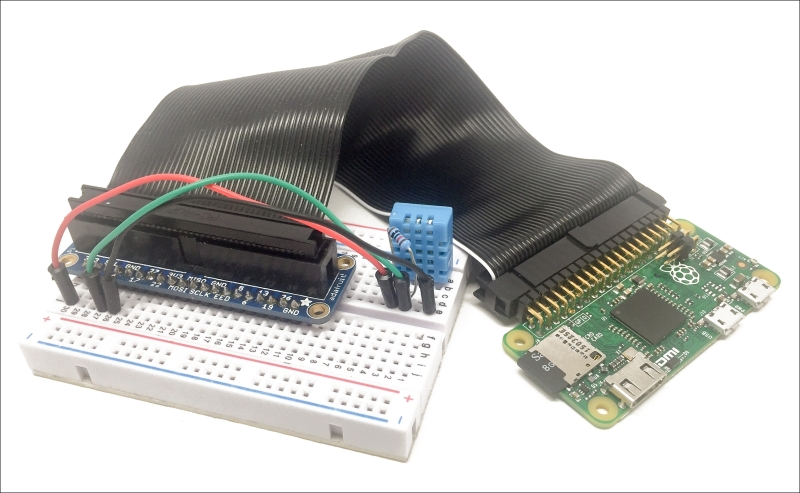
## Hardware configuration

Let's now look at how to configure the hardware for this project; basically, how to connect the sensor to the Pi Zero board.

The following figure is a schematic to help you out:



As it's the first project we are actually building using the Raspberry Pi Zero, there is something important I wanted to point out here. To connect the board to components like this sensor here, we have two options. You can either use jumper wires directly (as shown on the schematic), or use a cobbler kit to connect all the pins of the Pi to the breadboard, as shown in the following image:



This is up to you, and to be clear, I'll always show only the individual wires on the schematics, but use a cobbler kit to actually build the projects.

Here, you simply need to place the DHT11 on the breadboard, and then connect the resistor between the VCC and the data pins. Then, connect the VCC to the 3.3V pin of the Raspberry Pi, GND to GND, and finally, connect the data pin of the sensor to pin 4 of the Raspberry Pi board.

## Software configuration

Now we are going to install additional software on your Pi to make sure we can read data from the sensor.

Following the instructions from [Chapter 1](https://www.safaribooksonline.com/library/view/building-smart-homes/9781786466952/ch01.html), Configuring Your Raspberry Pi Zero Board log into your Pi via SSH, or just use it with an external screen with mouse and keyboard.

1. Inside a terminal, type the following:
2. **wget http://www.airspayce.com/mikem/bcm2835/bcm2835-1.50.tar.gz**
3. Wait for the download to complete and then type the following:
4. **tar zxvf bcm2835-1.50.tar.gz**
5. Next, type the following:
6. **cd bcm2835-1.50 b**
7. Now, configure the software you just downloaded with the following:
8. **./configure**
9. Build this software with the following:
10. **make**
11. Now, verify that everything is okay with the following:
12. **sudo make check**
13. If there are no errors, you can then install the software on your Pi with the following:
14. **sudo make install**

After that last step, you can now move on to the projects of this chapter!

# Reading data from the sensor

As the first project of this chapter, we are simply going to see how to read data from the sensor. As for all the projects in this book, we'll use Node.js, which is a great framework for building projects on your Raspberry Pi Zero.

I will now go through the main parts of this first piece of code. It starts by including the DHT sensor module for Node.js:

var sensorLib = require('node-dht-sensor');

Then, we create an object to read data from the sensor and initialize it when we start the software:

var sensor = {

initialize: function () {

return sensorLib.initialize(11, 4);

},

read: function () {

var readout = sensorLib.read();

console.log('Temperature: ' + readout.temperature.toFixed(2) + 'C, ' +

'humidity: ' + readout.humidity.toFixed(2) + '%');

setTimeout(function () {

sensor.read();

}, 2000);

}

};

if (sensor.initialize()) {

sensor.read();

} else {

console.warn('Failed to initialize sensor');

}

You can now either copy the code inside a file called sensor\_test.js, or just get the complete code from the GitHub repository for this project:

<https://github.com/openhomeautomation/smart-homes-pi-zero>

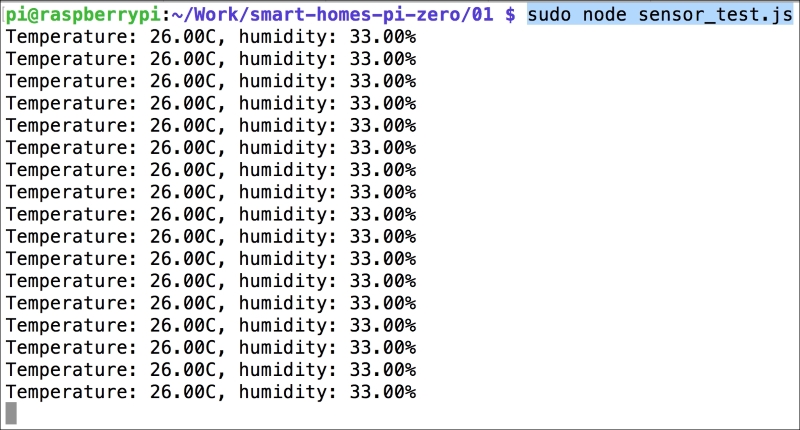
Next, use Terminal to navigate to the folder where the files are and type the following:

**npm install node-dht-sensor**

This will install the module to read data from the sensor; it can take a while, so be patient. In case it doesn't work, try using sudo in front of the command. Next, actually start the software with the following:

**sudo node sensor\_test.js**

This should print the readings of the sensor at regular intervals inside the terminal:



Congratulations, you can now read data from a digital sensor using your Pi Zero board! This is the first step to building sensors for your smart home.

# Storing sensor data

Displaying the current measurements from the sensor is nice, but what is even better is to actually store that data inside a database. In this section, we are going to see how easy it is to do this with Node.js.

As a database, we'll simply use NeDB here, which is a really simple database for Node.js that is completely stored in memory, but you can also save the entire database in a file.

The code is actually very similar to what we saw in the previous section. However, here, we'll first import the database module, and then insert data inside the database when a measurement is done:

var Datastore = require('nedb')

, db = new Datastore({ filename: 'path/to/datafile', autoload: true });

sdfsd

var readout = sensorLib.read();

// Log

var data = {

humidity: readout.humidity.toFixed(2),

temperature: readout.temperature.toFixed(2),

date: new Date()

};

db.insert(data, function (err, newDoc) {

console.log(newDoc);

});

// Repeat

setTimeout(function () {

sensor.read();

}, 2000);

You can of course find all the code inside the GitHub repository of the book. Again, navigate to the folder where the files are located and type the following:

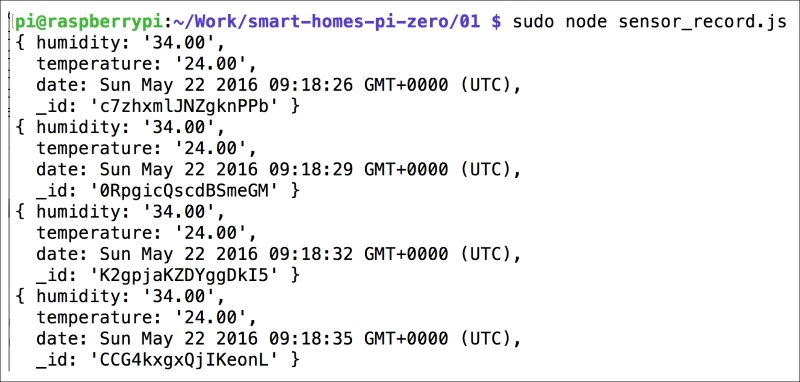
**npm install nedb --save**

This will install the NeDB module for Node.js.

Then, start the recording with the following:

**sudo node sensor\_record.js**

You should see the measurements being recorded at regular intervals:



Now, we didn't learn how to actually retrieve those measurements, but that's something we will see later in this chapter. In the meantime, if you want more information about how to retrieve documents, you can look at the official page on GitHub:

<https://github.com/louischatriot/nedb>

# Accessing the data remotely

In the previous projects of this chapter, we learned how to measure and store data on your Pi. However, in a smart home, the best is to be able to access data remotely, for example, from your smartphone or computer. We will see many similar examples in later chapters of this book, but in this chapter, I just wanted to give you a glimpse of what is possible.

The module we are going to use here is Express, a server framework that is really easy to use with Node.js. Express works by defining routes, which is what will be served to the client if a request is made on a specific URL.

First, we'll import Express and define a main route that will send back the temperature and humidity measurements:

var express = require('express');

var app = express();

app.get('/', function (req, res) {

var readout = sensor.read();

answer = 'Temperature: ' + readout.temperature.toFixed(2);

answer += ' Humidity: ' + readout.humidity.toFixed(2);

res.send(answer);

});

Finally, we also need to start the application, and once that's done we print a message in the console:

app.listen(3000, function () {

console.log('Raspberry Pi Zero app listening on port 3000!');

});

It's now time to test our little web server! Once you have grabbed the file from the book's GitHub repository, navigate to the folder where the files are and type the following:

**npm install express**

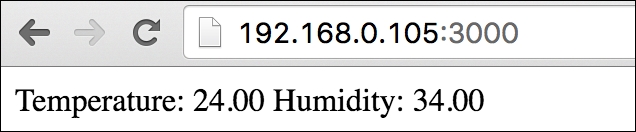
Then, launch the app with the following:

**sudo node sensor\_express.js**

You should get the confirmation inside the console. Now, using your computer or a smartphone, navigate to the URL of your Pi, not forgetting to add port 3000: http://192.168.0.105:3000/.

If you don't know the IP address of your Pi, you can simply type ifconfig while logged in.

The page should display the last measurement made by the Pi:



Of course, this can definitely be improved with a much nicer interface to display the measurements. However, in this section the goal was really to show you how to display those measurements from a device other than the Pi.

# Plotting the stored data

In the final project of this chapter, we are going to learn how to plot the data that was measured by the Raspberry Pi Zero board. We are actually going to combine what we did in the other projects of this chapter and add the plotting part on top of that.

As the code is quite similar to what we have already seen, I will only highlight the main changes here. First, we need to define a route for the data:

app.get('/data', function (req, res) {

db.find({}, function (err, docs) {

res.json(docs);

});

});

This will make sure that, when it is queried on this route, the server will return all the measurements stored so far inside the database.

Then, to display the plot of all the measurements, we are going to use a JavaScript called **HighCharts**. You can find more information about HighCharts here:

<http://www.highcharts.com/>

We'll include it inside an HTML file that we will place inside a folder called public, so our app can access it. This file will basically import all the JavaScript libraries that we need, another script, which we'll see in a moment, and a container for the plot:

<!DOCTYPE html>

<html>

<head>

<script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>

<script src="https://code.highcharts.com/highcharts.js"></script>

<script src="https://code.highcharts.com/modules/exporting.js"></script>

<script src="js/script.js"></script>

</head>

<body>

<div id="container" style="min-width: 310px; height: 400px; margin: 0 auto"></div>

</body>

</html>

Now, we also need to make the link between the HTML page and our application. This will be done in a script file called script.js. This is the content of this file:

var dates = [];

var temperature = [];

var humidity = [];

console.log(measurements);

for (i = 0; i < measurements.length; i++) {

dates.push(measurements[i].date);

temperature.push(parseFloat(measurements[i].temperature));

humidity.push(parseFloat(measurements[i].humidity));

}

$('#container').highcharts({

title: {

text: 'Temperature & Humidity Data',

x: -20 //center

},

xAxis: {

categories: dates

},

yAxis: {

title: {

text: 'Temperature (°C)'

},

plotLines: [{

value: 0,

width: 1,

color: '#808080'

}]

},

tooltip: {

valueSuffix: '°C'

},

legend: {

layout: 'vertical',

align: 'right',

verticalAlign: 'middle',

borderWidth: 0

},

series: [{

name: 'Temperature',

data: temperature

},

{

name: 'Humidity',

data: humidity

}]

});

Basically, this file will query the most recent data from the application, format it for HighCharts, and then actually plot the data.

You can now grab all the files from the book's GitHub repository and start the application with the following:

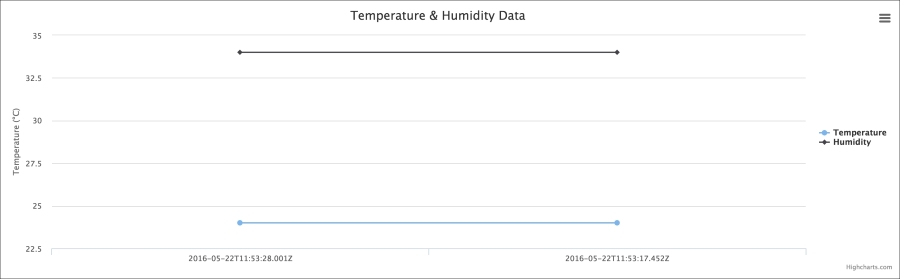
**sudo node sensor\_plot.js**

The first thing you can do is test the data route by going to the IP address of your Pi, followed by port 3000 and the data route:

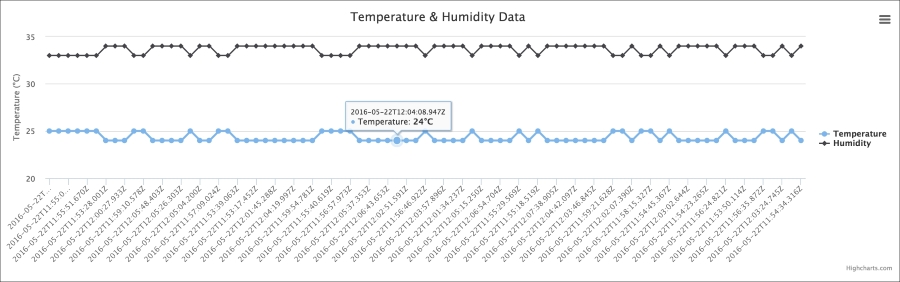
Plotting the stored data

As you can see, all the measurements made so far are extracted from the database and returned by the server.

You can again go to the main route, and you should see the same data on a nice plot:



If you wait a bit more, you'll see a nice graph with all the data recorded so far by the application running on your Raspberry Pi board:



Of course, you can adjust some settings, for example, the delay between two measurements, inside the code for your own projects.

**Chapter 6. Sending Notifications using Raspberry Pi Zero**

In this chapter, we are going to start diving into a very interesting field that will change the way we interact with our environment: the **Internet of Things** (**IoT**). The IoT basically proposes to connect every device around us to the Internet, so we can interact with them from anywhere in the world.

Within this context, a very important application is to receive notifications from your devices when they detect something in your home, for example a motion in your home or the current temperature. This is exactly what we are going to do in this chapter, we are going to learn how to make your Raspberry Pi Zero board send you notifications via text message, email, and push notifications. Let's start!

**Hardware and software requirements**

As always, we are going to start with the list of required hardware and software components for the project.

Except Raspberry Pi Zero, you will need some additional components for each of the sections in this chapter.

For the first project of this chapter, we are going to use a simple PIR motion sensor to detect motion from your Pi.

Then, for the last two projects of the chapter, we'll use the DHT11 sensor that we have already used in previous chapters.

Finally, you will need the usual breadboard and jumper wires.

This is the list of components that you will need for this whole chapter, not including the Raspberry Pi Zero:

* PIR motion sensor (<https://www.sparkfun.com/products/13285>)
* DHT11 sensor + 4.7k Ohm resistor (<https://www.adafruit.com/products/386>)
* Breadboard (<https://www.adafruit.com/products/64>)
* Jumper wires (<https://www.adafruit.com/products/1957>)

On the software side, you will need to create an account on IFTTT, which we will use in all the projects of this chapter. For that, simply go to:

<https://ifttt.com/>

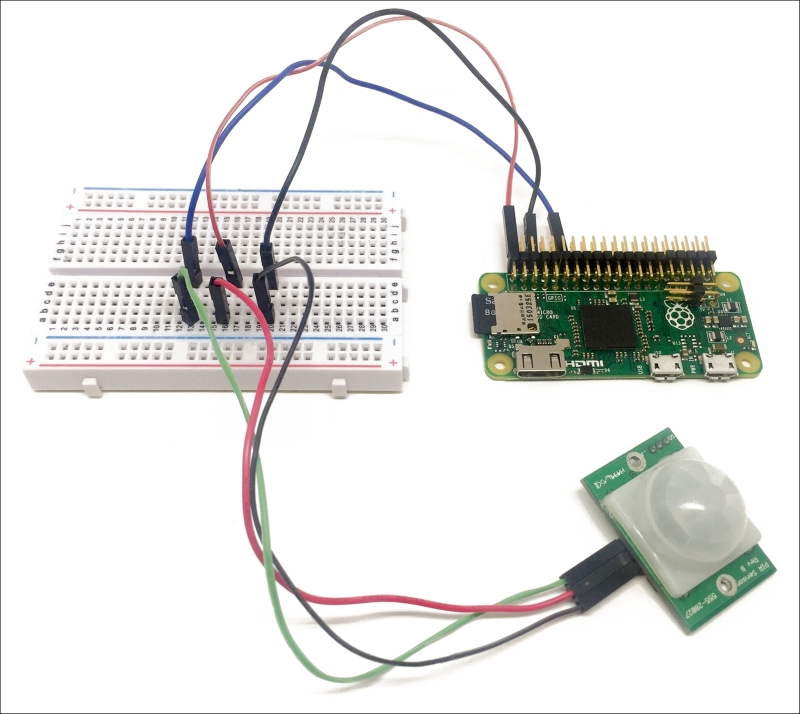
You should be redirected to the main page of IFTTT where you'll be able to create an account:

# Making a motion sensor that sends text messages

For the first project of this chapter, we are going to attach a motion sensor to the Raspberry Pi board and make the Raspberry Pi Zero send us a text message whenever motion is detected. For that, we are going to use IFTTT to make the link between our Raspberry Pi and our phone. Indeed, whenever IFTTT will receive a trigger from the Raspberry Pi, it will automatically send us a text message.

Lets first connect the PIR motion sensor to the Raspberry Pi. For that, simply connect the VCC pin of the sensor to a 3.3V pin of the Raspberry Pi, GND to GND, and the OUT pin of the sensor to GPIO18 of the Raspberry Pi.

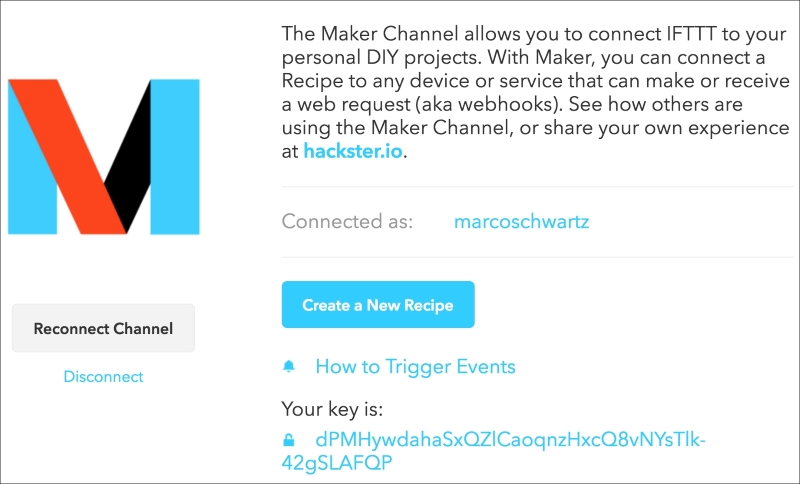
This is the final result:



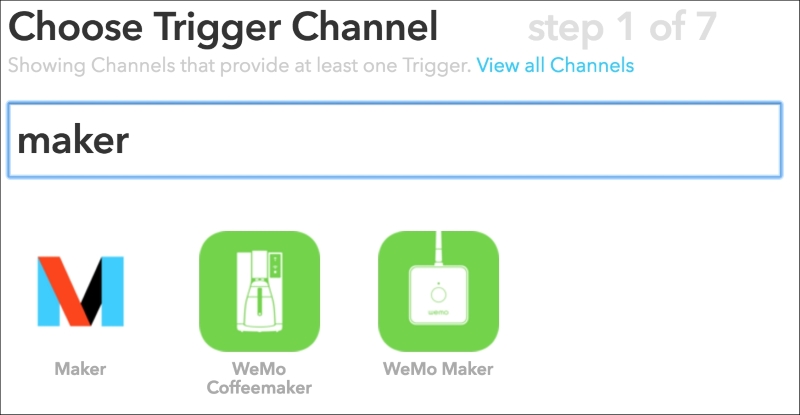
Let's now add our first channel to IFTTT, which will allow us later to interact with the Raspberry Pi and with web services. You can easily add new channels by clicking on the corresponding tab on the IFTTT website. First, add the **Maker** channel to your account:



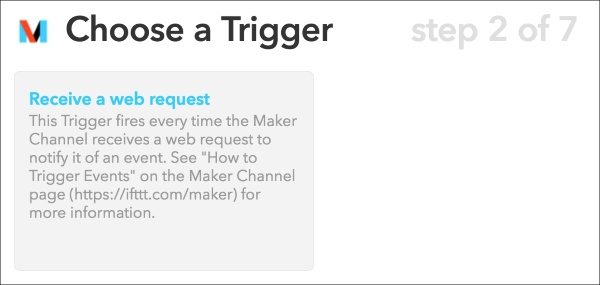
This will basically give you a key that you will need when writing the code for this project:



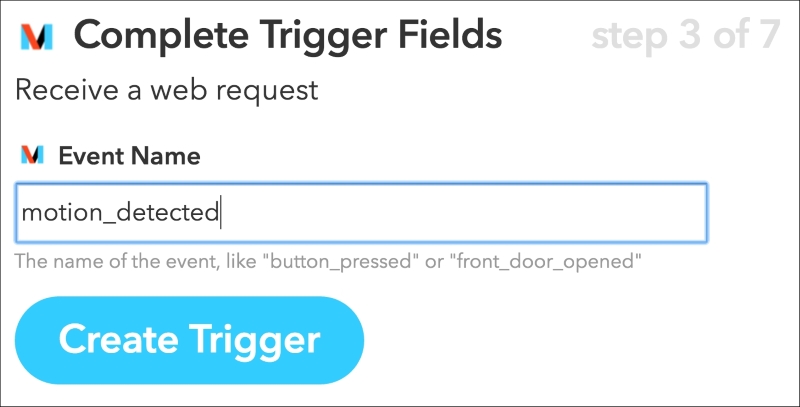
After that, add the **SMS** channel to your IFTTT account. Now, you can actually create your first recipe. Select the Maker channel as the trigger channel:



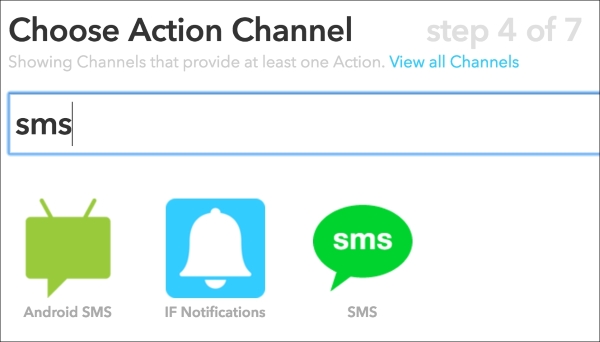
Then, select **Receive a web request**:



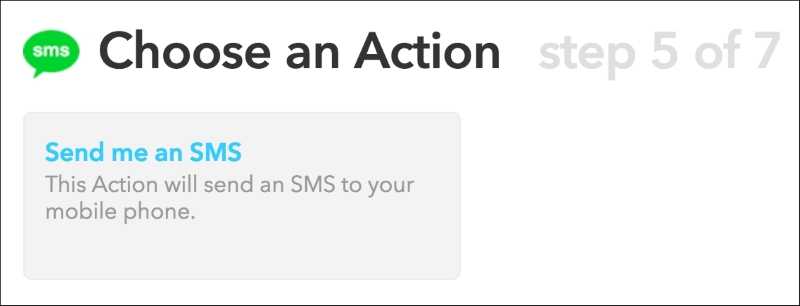
As the name of this request, enter motion\_detected:



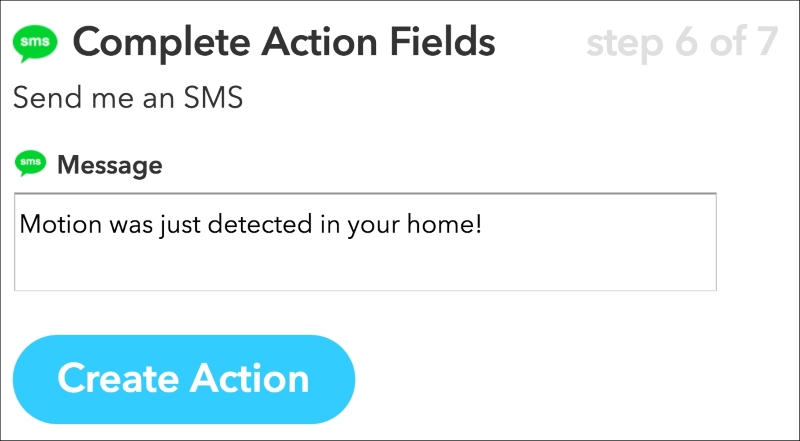
As the action channel, which is the channel that will be executed when a trigger is received, choose the SMS channel:



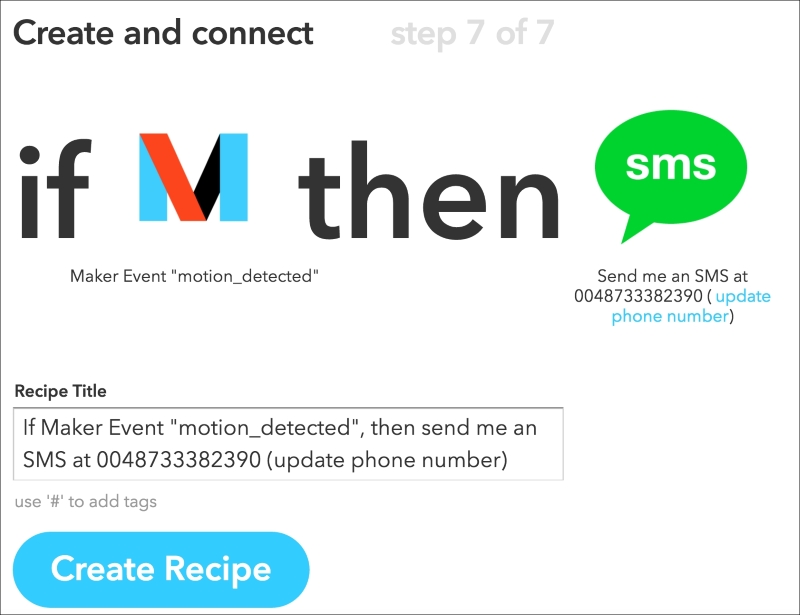
For the action, choose **Send me an SMS**:



You can now enter the message you want to see in the text messages:



Finally, confirm the creation of the recipe:



Now that our recipe is created and active, we can move on to actually configuring Raspberry Pi so it sends alerts whenever a motion is detected. As usual, we'll use Node.js to code this program.

It starts by including the required modules:

// Required modules

var request = require('request');

var gpio = require('rpi-gpio');

Then, we define our IFTTT data, which is composed of the Maker key and of the name of the event we want to trigger:

// IFTTT data

var key = "your-key";

var eventName = 'motion\_detected';

Then, we define the pin on which our sensor is connected to:

// Motion sensor GPIO

var motionSensorPin = 18;

We also need to define a counter that will basically make sure that we don't constantly send alerts to our phone, for example if the sensor stays on for several seconds. For that, we define an interval period of one minute minimum between two alerts:

// Counter between two alerts

var interval = 60 \* 1000; // 1 minute

var counter = new Date();

After that, we configure the rpi-gpio module (that we'll use to read data from the sensor) to the BCM configuration scheme, meaning we are using the number of the GPIO of the Raspberry Pi rather than the physical pins:

// Setup gpio library

gpio.setMode(gpio.MODE\_BCM);

Now, every second, we check the status of the sensor (if we didn't trigger an alert already in the last minute):

// Check status every second

setInterval(function() {

// Check counter so we don't trigger alarms all the time

var currentTime = (new Date()).getTime();

var counterTime = counter.getTime();

if ( (currentTime - counterTime) > interval) {

// Check sensor

gpio.setup(motionSensorPin, gpio.DIR\_IN, checkSensor);

}

}, 1000);

If some motion is detected, we reset the counter and also send the alert to IFTTT:

// Check motion sensor

function checkSensor() {

gpio.read(motionSensorPin, function(err, value) {

// If motion is detected, send event to IFTTT

if (value == true) {

// Restart Counter

counter = new Date();

// Send event

alertIFTTT();

}

});

}

This is the function that takes care of sending the data to IFTTT:

// Make request

function alertIFTTT() {

// Send alert to IFTTT

console.log("Sending alert to IFTTT");

var url = 'https://maker.ifttt.com/trigger/' + eventName + '/with/key/' + key;

request(url, function (error, response, body) {

if (!error && response.statusCode == 200) {

console.log("Alert sent to IFTTT");

}

});

}

It basically uses the request module to send the correct command to IFTTT, passing the name of the event and the key.

We can finally test this first project! Grab the code from the GitHub repository of the book and make sure to modify the code with your own IFTTT key. Then, navigate to the folder where the file is with a terminal and type the following command:

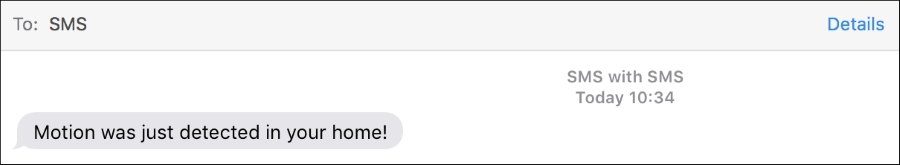
**sudo npm install rpi-gpio request**

Once that's done, start the code with:

**sudo node sms\_alerts.js**

Now, you need to wait at least for the interval time (one minute by default) before the code is active. This will make sure that no motion will be detected when you just start the project for example.

After a minute, pass your hand in front of the sensor: Your Raspberry Pi should immediately send a command to IFTTT and after some seconds you should be able to receive a message on your mobile phone:



Congratulations, you can now use your Raspberry Pi Zero to send important notifications, on your mobile phone!

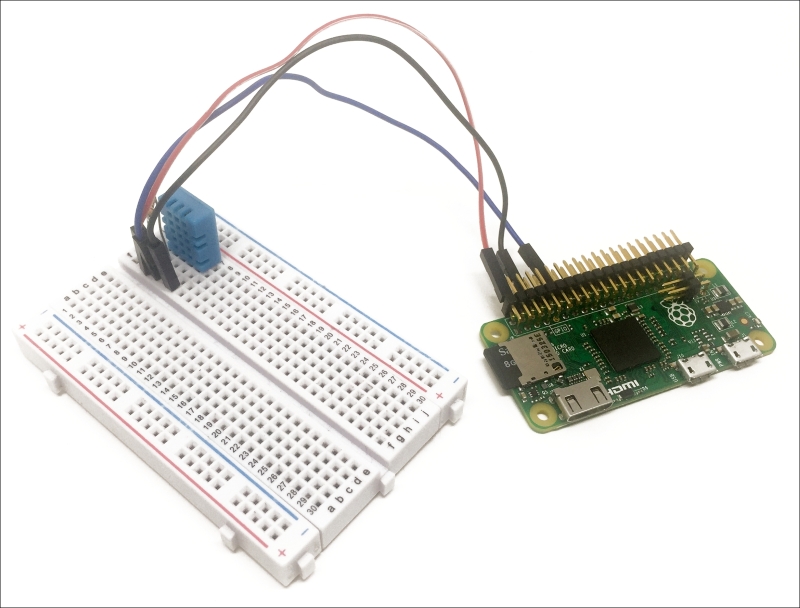
### NOTE

Note that for an actual use of this project in your home, you might want to limit the number of messages you are sending as IFTTT has a limit on the number of messages you can send (check the IFTTT website for the current limit). For example, you could use this for only very important alerts, like in case of an intruder coming in your home in your absence.

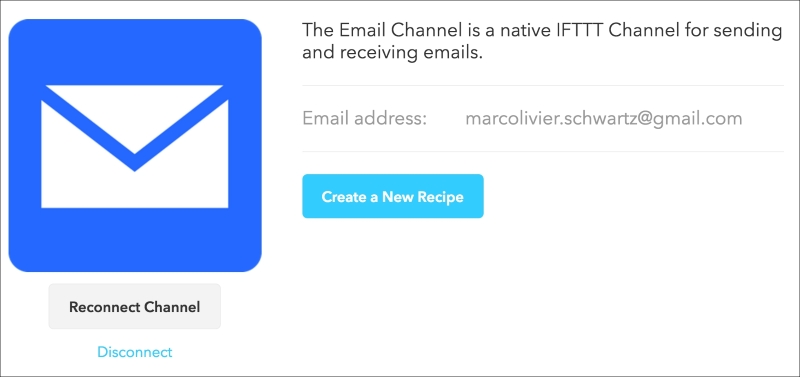
# Sending temperature alerts through email

In the second project of the chapter, we are going to learn how to send automated email alerts based on data measured by the Raspberry Pi.

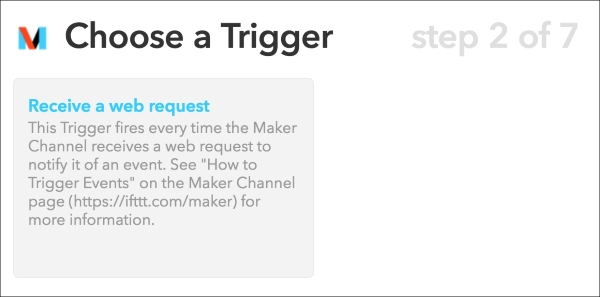
Let's first assemble the project. Place the DHT11 sensor on the breadboard and then place the 4.7k Ohm resistor between pin 1 and 2 of the sensor. Then, connect pin 1 of the sensor to the 3.3V pin of the Raspberry Pi, pin 2 to GPIO18, and pin 4 to GND. This is the final result:



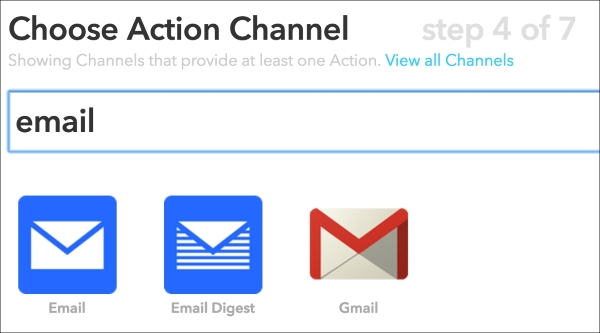
Let us now see how to configure the project. Go over to IFTTT and create add the **Email Channel** to your account:



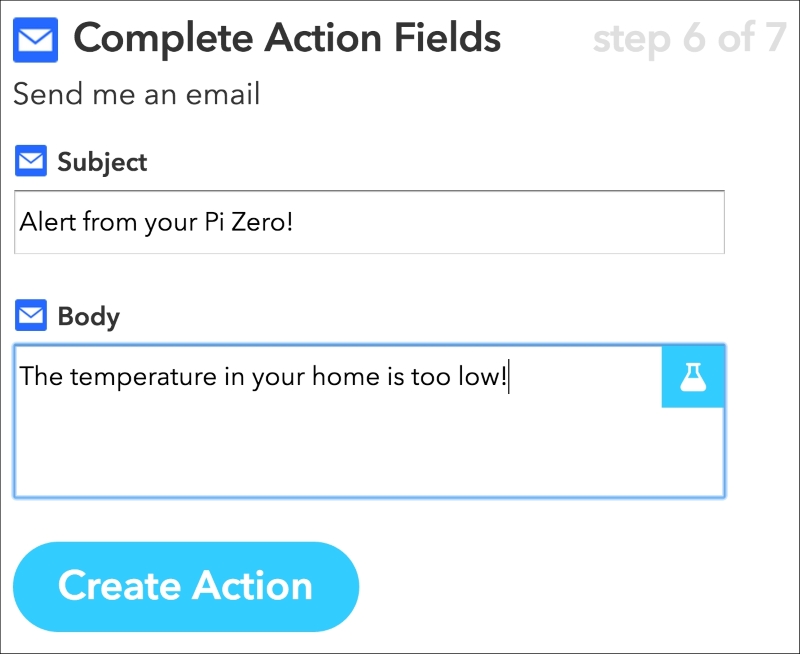
After that, create a new recipe by choosing the Maker channel as the trigger:



For the event, enter temperature\_alert and then choose **Email** as the action channel:



You will then be able to customize the text and subject of the email sent to Pi. As we want to send the emails whenever the temperature in your home gets too low, you can use a similar message:



You can now finalize the creation of the recipe and close IFTTT. Let's now see how to configure the Raspberry Pi Zero. As the code for this project is quite similar to the one we saw in the previous section, I will only highlight the main differences here.

It starts by including the required components:

var request = require('request');

var sensorLib = require('node-dht-sensor');

Then, give the correct name to the event we'll use in the project:

var eventName = 'temperature\_low';

We also define the pin on which the sensor is connected:

var sensorPin = 18;

As we want to send alerts based on the measured temperature, we need to define a threshold. As it was quite warm when I made this project, I have assigned a high threshold at 30 degrees Celsius, but you can, of course, modify it:

var threshold = 30;

Then, we initialize the sensor and check the current temperature every 2 seconds:

var sensor = {

initialize: function () {

return sensorLib.initialize(11, sensorPin);

},

read: function () {

// Read

var readout = sensorLib.read();

temperature = readout.temperature.toFixed(2);

console.log('Current temperature: ' + temperature);

// Check counter so we don't trigger IFTTT all the time

var currentTime = (new Date()).getTime();

var counterTime = counter.getTime();

if ( (currentTime - counterTime) > interval) {

if (temperature < threshold) {

// Restart Counter

counter = new Date();

// Send event

alertIFTTT();

}

}

// Repeat

setTimeout(function () {

sensor.read();

}, 2000);

}

};

It's now time to test the project! Grab the GitHub repository of the book and make sure to modify the code to put your own IFTTT key.

Then, navigate to the folder with the file with a terminal and type the following:

**sudo npm install request**

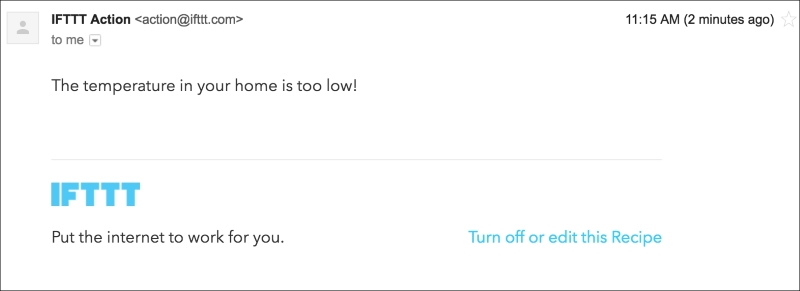
Then, install the module for the DHT sensor with the following command:

**npm install node-dht-sensor**

Finally, start the project using the following command:

**sudo node temperature\_alerts.js**

If like me, you set the threshold to a quite high value, you should quickly receive a message inside your email box:



You can now use your Pi to send automated notifications through email!

# Receiving measurement SATA through push notifications

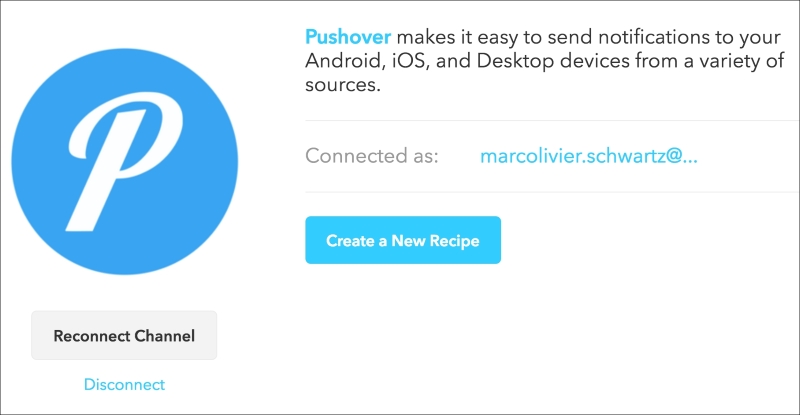
In the last project of this chapter, we'll learn how to use the project we built in the previous section to actually not send you alerts, but just keep you updated about the current temperature and humidity measured by Pi.

Here however, we are going to use something new to alert you: push notifications. These notifications will immediately show up on your phone if you have the right app installed.

As the app, we'll use Pushover that is available for iOS and Android. You can install it from your App Store and find more information at the following URL:

<https://ifttt.com/>

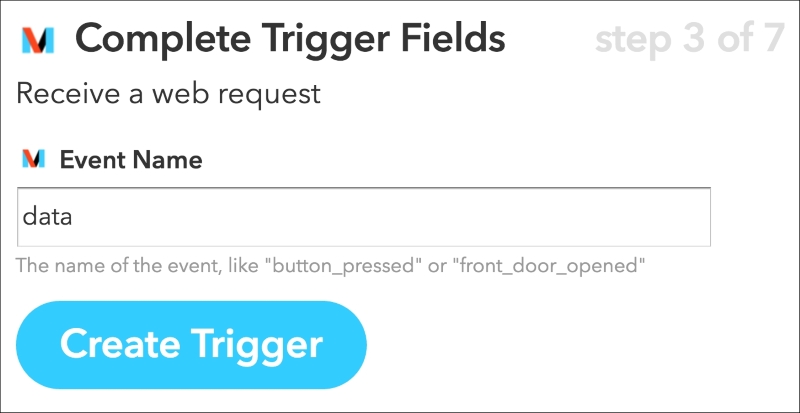
Then, add the **Pushover** channel inside IFTTT:



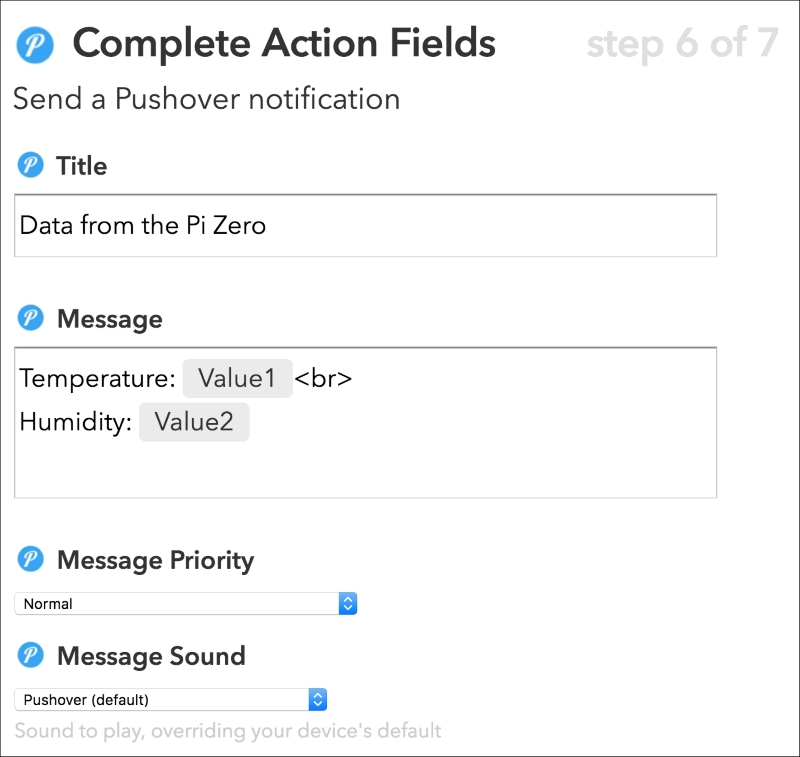
Now, create a new recipe and choose the **Maker** channel again as the trigger:



I used **data** as the trigger:

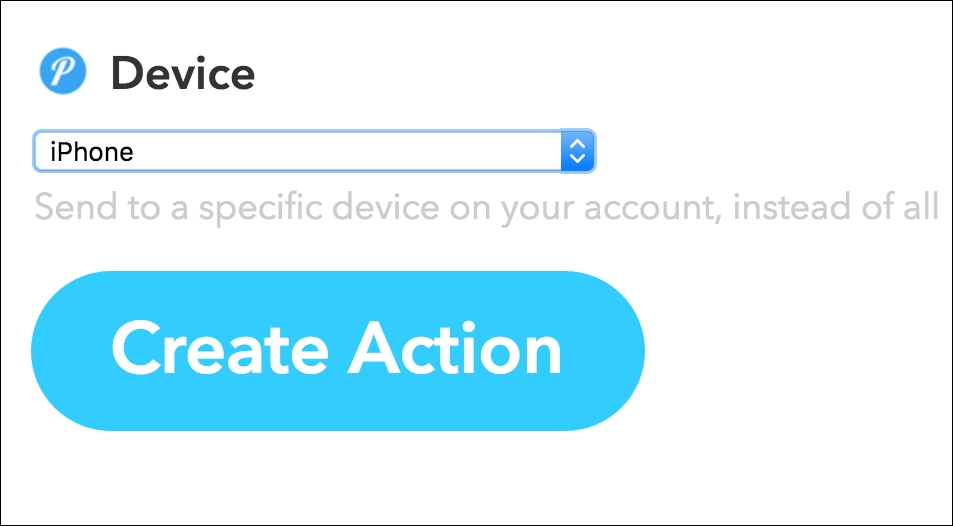


Then, select **Pushover** as the action channel and enter the following message:



Here, we'll use the variables **Value1** and **Value2** to display the temperature and humidity inside the message. We'll see in a moment how to actually send that to IFTTT from the Pi.

You can also select the target device for this recipe:



Lets now see how to configure this project. As it's really similar to the previous project, I will just highlight the main differences here.

We need to give the name to the event that we'll trigger from the Pi:

var eventName = 'data';

Then, inside the sensor object we measure data at regular intervals and pass the measurements to the logIFTTT()function:

var sensor = {

initialize: function () {

return sensorLib.initialize(11, sensorPin);

},

read: function () {

// Read

var readout = sensorLib.read();

temperature = readout.temperature.toFixed(2);

humidity = readout.humidity.toFixed(2);

console.log('Current temperature: ' + temperature);

console.log('Current humidity: ' + humidity);

// Send event

logIFTTT(temperature, humidity);

// Repeat

setTimeout(function () {

sensor.read();

}, interval);

}

};

Let's now see the details of this function. Compared to the previous projects of this chapter, we are now going to pass the temperature and humidity parameters to the function and send this data to IFTTT:

function logIFTTT(temperature, humidity) {

// Send alert to IFTTT

console.log("Sending message to IFTTT");

var url = 'https://maker.ifttt.com/trigger/' + eventName + '/with/key/' + key;

url += '?value1=' + temperature + '&value2=' + humidity;

request(url, function (error, response, body) {

if (!error && response.statusCode == 200) {

console.log("Data sent to IFTTT");

}

});

}

Let's now test the project! First, grab all the code from the GitHub repository of the book and make sure to modify the code with your IFTTT maker key. Then, install the request module with the following command:

**sudo npm install request**

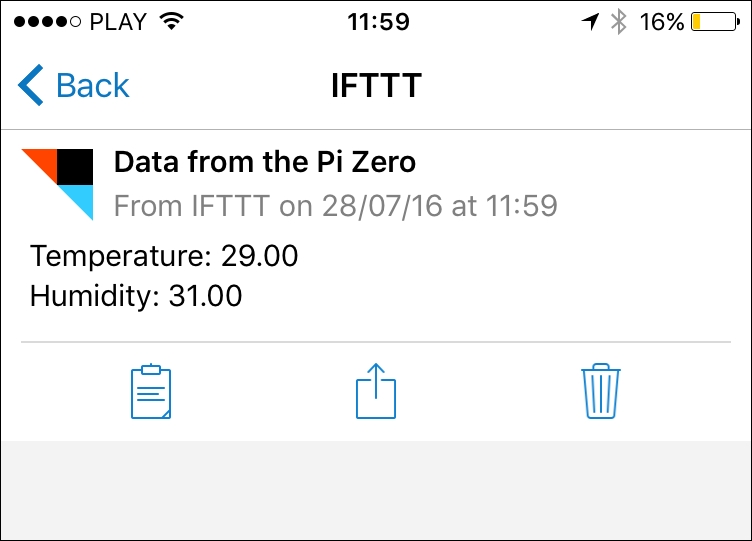
Once that's done, install the module for the DHT sensor:

**npm install node-dht-sensor**

Finally, start the project using this command:

**sudo node temperature\_notifications.js**

After a minute, you should get the notification on your phone displaying the current temperature and humidity in your home:



You can now use your Pi to receive automated reports containing data about your home!

**Chapter 7. Use the Raspberry Pi Zero to Build a Security System**

In this chapter, we are going to learn how to build a modular security system using the Raspberry Pi Zero board. The Raspberry Pi board is really cheap and has a very small form factor, you can use many such boards inside your home to build a complete security system for your home.

We are going to integrate three types of components into our system: motion sensors, alarms, and security camera. These modules will communicate with a central server application that will either run on your computer or on another Raspberry Pi. First, we are going to see how to configure each board individually and then configure the central server and a basic interface.

**Hardware and software requirements**

As always, we are going to start with the list of required hardware and software components for the project.

In this chapter, we are going to use at least three Raspberry Pi Zero boards: for a motion sensor, an alarm module, and a camera module. Of course, you can perfectly use more of each module in your security system.

For the motion sensor module, I will use a simple PIR motion sensor.

Then, for the alarm module, I will be using a small buzzer, as well as an LED and a 330 Ohm resistor.

For the camera module, I will use a Logitech C270 webcam. Here, any camera compatible with the UVC protocol would work, which is the case for most of the cameras sold these days.

Finally, you will need the usual breadboard and jumper wires.

This is the list of components that you will need for this chapter, not including the Raspberry Pi Zero:

* PIR motion sensor (<https://www.sparkfun.com/products/13285>)
* LED (<https://www.sparkfun.com/products/9590>)
* 330 Ohm resistor (<https://www.sparkfun.com/products/11507>)
* Logitech C270 USB camera (<http://www.logitech.com/en-us/product/hd-webcam-c270>)
* Breadboard (<https://www.adafruit.com/products/64>)
* Jumper wires (<https://www.adafruit.com/products/1957>)

Of course, all the additional components, for example the WiFi dongle and power supply, will need to be multiplied by the number of Raspberry Pi boards that you will use inside the project.

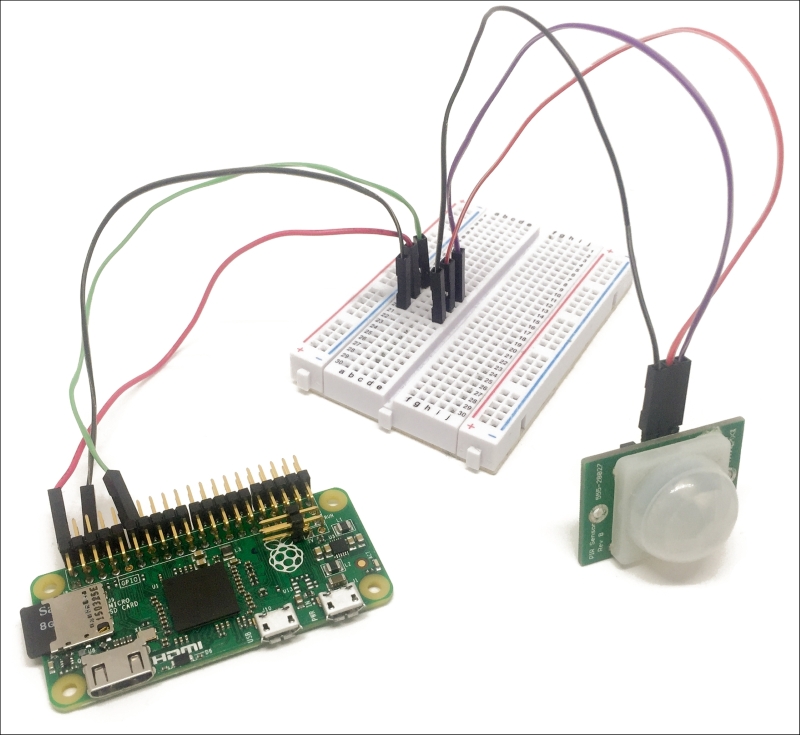
On the software side, you will just need to have Node.js installed on your Raspberry Pi Zero boards.

# Building a motion sensor with the Pi Zero

The first module that we are going to assemble in this chapter is the motion sensor module. These modules will be deployed in key parts of your home, to detect any intruder in your home.

The hardware configuration for this part will actually be very simple. First, connect the VCC pin of the motion sensor to a 3.3V pin of the Raspberry Pi. Then, connect the GND pin of the sensor to one GND pin of the Pi. Finally, connect the OUT pin of the motion sensor to the GPIO17 pin of the Raspberry Pi. You can refer to the previous chapters to find out about pin mapping of the Raspberry Pi Zero board.

This is the final result:



Let's now see how to configure this module so we can access it remotely through WiFi. This application will be based on the **aREST framework** again, which we already saw in the previous chapters of the book.

Here is the complete code for this part:

// Modules

var express = require('express');

// Express app

var app = express();

// aREST

var piREST = require('pi-arest')(app);

piREST.set\_id('34f5eQ');

piREST.set\_name('motion\_sensor');

piREST.set\_mode('bcm');

// Start server

app.listen(3000, function () {

console.log('Raspberry Pi Zero motion sensor started!');

});

You can now simply grab this code from the GitHub repository of the book or simply paste it into a file called motion\_sensor.js, then using a terminal inside the same folder as the file type:

**sudo npm install express pi-arest**

Once the required modules are installed, type the following command to start the project:

**sudo node motion\_sensor.js**

Finally, navigate to the IP address of your Pi on port 3000, followed by the digital command on pin 17:

**http://192.168.0.105:3000/digital/17**

This should immediately return a JSON object with the value of pin 17. You can now try to pass your hand in front of the sensor and repeat the operation: you should be able to immediately see that the value of pin 17 is equal to 1, indicating that motion has been detected by the sensor.

# Making a simple alarm module

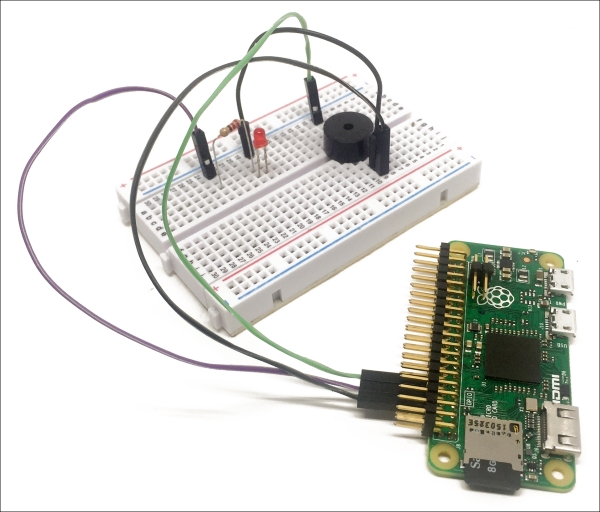
In the second part of this chapter, we are going to learn how to build an alarm module for our security system. You will usually have one of those modules in your home that will flash light and emit sound in case motion is detected. Of course, you can perfectly connect it to a real siren instead of a buzzer to have a loud sound in case any motion is detected.

To assemble this module, first place the LED in series with the 330 Ohm resistor on the breadboard, with the longest pin of the LED in contact with the resistor. Also place the buzzer on the breadboard.

Then, connect the other side of the resistor to GPIO14 of the Pi and the other part of the LED to one GND pin of the Pi.

For the buzzer, connect the pin marked as **+** on the buzzer to GPIO15 and the other pin of the buzzer to one GND pin of the Pi.

This is the final result:



To configure this module, we will again use the aREST library, so the code will be very similar to the one we used in the previous section:

// Modules

var express = require('express');

// Express app

var app = express();

// aREST

var piREST = require('pi-arest')(app);

piREST.set\_id('35f5fc');

piREST.set\_name('alarm');

piREST.set\_mode('bcm');

// Start server

app.listen(3000, function () {

console.log('Raspberry Pi Zero alarm started!');

});

You can now simply grab this code from the GitHub repository of the book or simply paste it into a file called alarm.js using a terminal inside the same folder as the file type:

**sudo npm install express pi-arest**

Once the required modules are installed, type the following command to start the project:

**sudo node alarm.js**

Finally, let's just try to set the buzzer on; navigate to the IP address of your Pi on port 3000 followed by the digital command on pin 15:

http://192.168.0.105:3000/digital/17/1

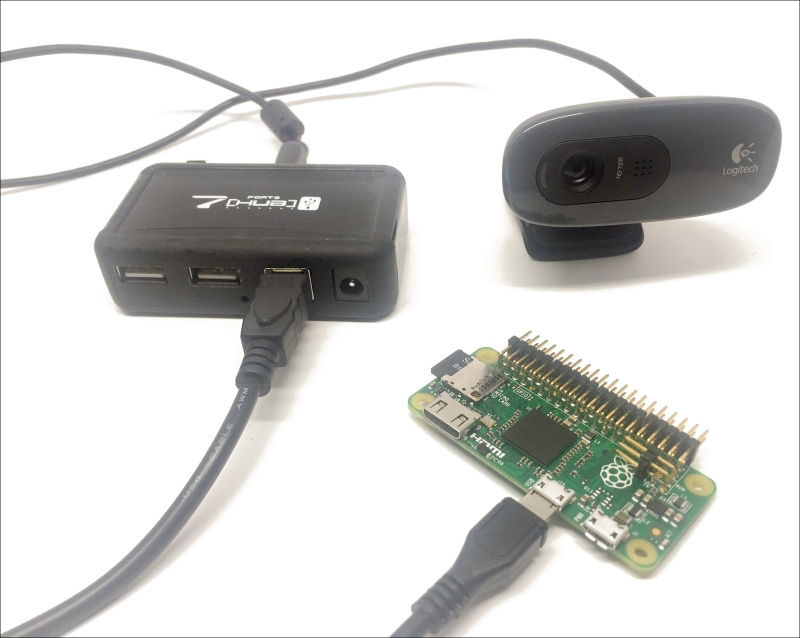
This should immediately set the buzzer on and it should continuously emit sound. To switch it off again, simply type the same command followed by a 0.

# Building a wireless security camera

We are now going to build the module that will act as a wireless security camera. You can have one or many of those modules inside your home; it will allow you to observe what is going on in your home from a central location.

The hardware configuration for this part will be really simple, as we are using an USB camera. However, you will need to use an USB hub here, as we will need to connect the USB camera and the usual WiFi dongle on the Raspberry Pi.

This is the final result:



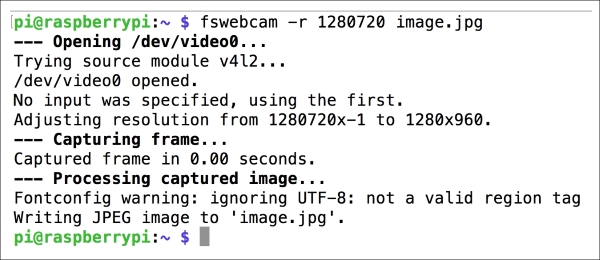
Let's now test the camera first, by taking a simple picture from the command line. You will need to install the fswebcam utility. To do so, simply type the following command inside a terminal:

**sudo apt-get install fswebcam**

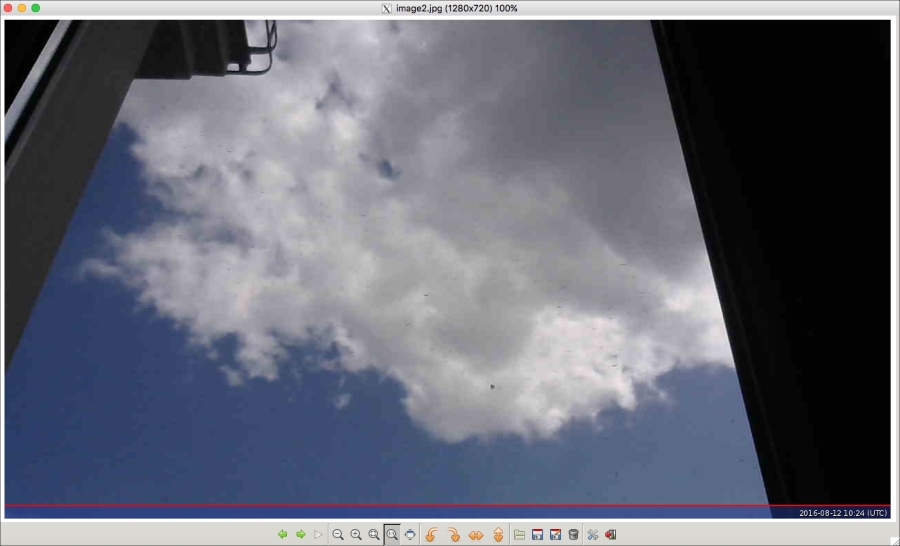
Then, still from a terminal, you can take a picture with the following command:

**fswebcam -r 1280×720 image.jpg**

This will make a lot of messages appear inside the terminal, confirming that the picture has been taken:



You can now use an image utility to open the picture you just took. I, for example, used GPicView:



Now that we are sure that the camera is working correctly, we can use it to stream video on the network. For that, we'll use software called MJPG-streamer. To install it, first clone the GitHub repository from a terminal:

**git clone https://github.com/jacksonliam/mjpg-streamer**

Then, install some required packages:

**sudo apt-get install cmake libjpeg62-dev**

Once that's done, navigate into the folder of the mjpg-streamersoftware and type:

**sudo make clean all**

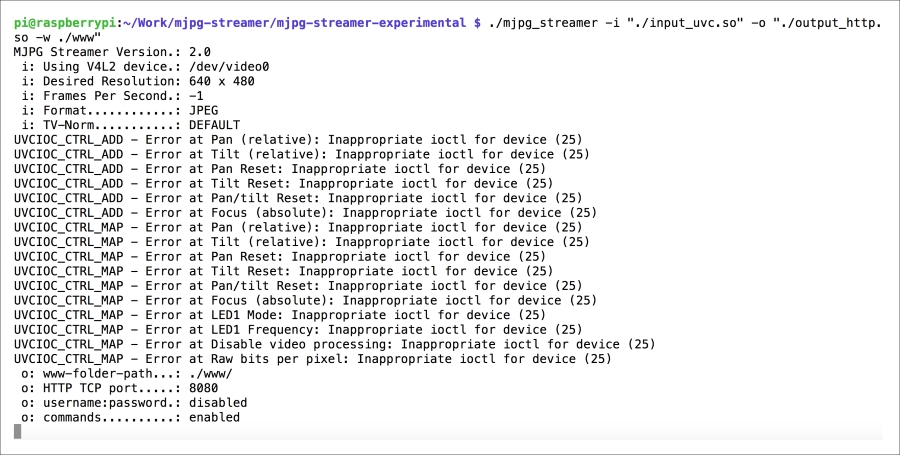
When the compilation of the software is done, type:

**export LD\_LIBRARY\_PATH=.**

Finally, start the software with the following command:

**./mjpg\_streamer -i "./input\_uvc.so" -o "./output\_http.so -w ./www"**

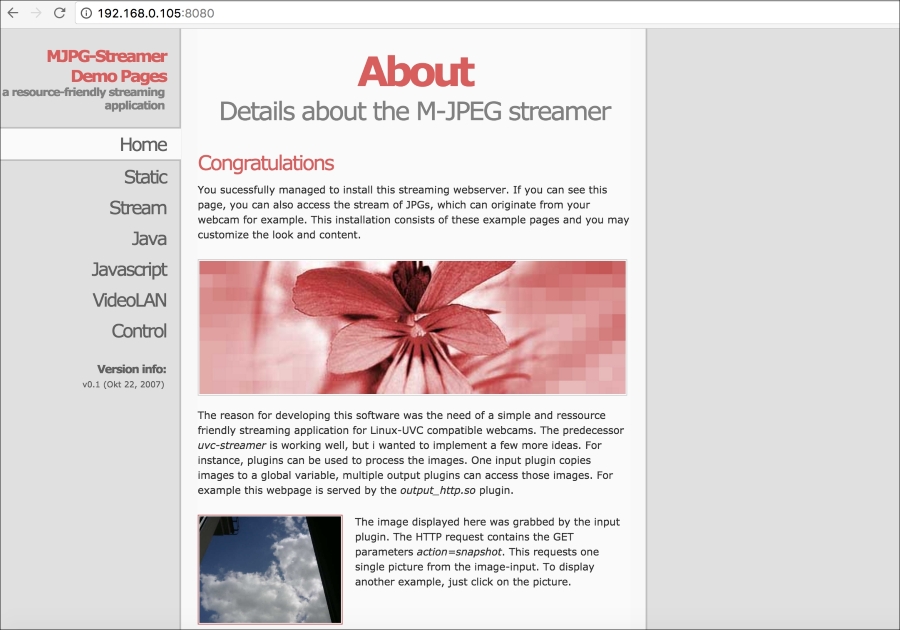
You should be able to see a similar output inside the terminal:



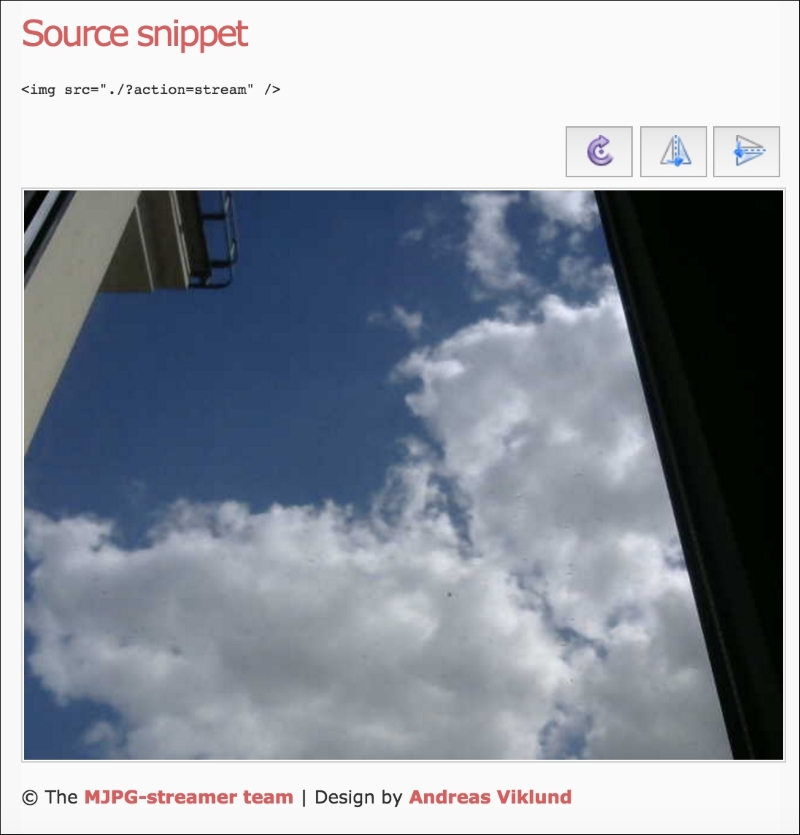
Then, simply navigate to port 8080 of your Raspberry Pi to access the interface of the streaming software, for example:

http://192.168.0.105:8080

You should be able to see the interface:



You can now just click on **Stream** to access the live stream from the camera:



From there, you can monitor the live stream from the camera. In the next section, we are going to see how to integrate this stream (and streams from other cameras if you have many) into a central interface.

### NOTE

Note that this project would also work with the official Raspberry Pi camera. However, the first versions of the Raspberry Pi Zero board didn't have the connector for the Raspberry Pi camera, so first make sure if your board has this connector (which is on the side of the board). For more information, visit this link:

<https://www.raspberrypi.org/blog/zero-grows-camera-connector/>

# Creating a security system

In the last section of this chapter, we are going to learn how to integrate all the modules we built in this chapter into a central interface, from which you'll be able to monitor them.

For this project, I ran this last part on my personal computer, but you can, of course, use another Pi Zero board (or any Raspberry Pi board) to run this software.

Let's now see the code for this last section. It will be again composed a main Node.js file for the server, and one HTML and JavaScript files for the interface itself.

Let's first see the Node.js part. It starts by importing all the required modules:

// Modules

var express = require('express');

var app = express();

var request = require('request');

// Use public directory

app.use(express.static('public'));

Then, you will need to modify the code to put the IP addresses of the Raspberry modules you will be using in the project (except the camera modules, we'll set their IPs directly inside the interface):

var motionSensorPi = "192.168.0.104:3000";

var alarmPi = "192.168.0.103:3000"

We also define the pins of the different components that are connected to our modules:

var buzzerPin = 15;

var ledPin = 14;

var motionSensorPin = 17;

Then, we can define the different routes of the project. It starts with a main route that will serve the interface:

app.get('/', function (req, res) {

res.sendfile(\_\_dirname + '/public/interface.html');

});

We also need to define a route to get the current state of the alarm:

app.get('/alarm', function (req, res) {

res.json({alarm: alarm});

});

Then, we set another route to set the alarm off:

app.get('/off', function (req, res) {

// Set alarm off

alarm = false;

// Set LED & buzzer off

request("http://" + alarmPi + "/digital/" + ledPin + '/0');

request("http://" + alarmPi + "/digital/" + buzzerPin + '/0');

// Answer

res.json({message: "Alarm off"});

});

We also start the server itself:

var server = app.listen(3000, function() {

console.log('Listening on port %d', server.address().port);

});

Finally, we create a loop that will check the status of the motion sensor every two seconds and set the alarm if motion is detected:

setInterval(function() {

// Get data from motion sensor

request("http://" + motionSensorPi + "/digital/" + motionSensorPin,

function (error, response, body) {

if (!error && body.return\_value == 1) {

// Activate alarm

alarm = true;

// Set LED on

request("http://" + alarmPi + "/digital/" + ledPin + '/1');

// Set buzzer on

request("http://" + alarmPi + "/digital/" + buzzerPin + '/1');

}

});

}, 2000);

Let's now see the interface file, starting by the HTML. It starts by importing all the required libraries and files for the project:

<head>

<script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"></script>

<script src="js/script.js"></script>

<link rel="stylesheet" href="css/style.css">

<meta name="viewport" content="width=device-width, initial-scale=1">

</head>

Then, inside a <script> tag on the same page, we'll define some JavaScript functions to integrate the live video stream into the page. We start by declaring the required variables:

var imageNr = 0; // Serial number of current image

var finished = new Array(); // References to img objects which have finished downloading

var paused = false;

Then, we declare a function that will create the image layer on the page that will later display the video server:

function createImageLayer() {

var img = new Image();

img.style.position = "absolute";

img.style.zIndex = -1;

img.onload = imageOnload;

img.onclick = imageOnclick;

img.src = "http://192.168.0.105:8080/?action=snapshot&n=" + (++imageNr);

var webcam = document.getElementById("webcam");

webcam.insertBefore(img, webcam.firstChild);

}

We then define a function to load the next image:

function imageOnload() {

this.style.zIndex = imageNr; // Image finished, bring to front!

while (1 < finished.length) {

var del = finished.shift(); // Delete old image(s) from document

del.parentNode.removeChild(del);

}

finished.push(this);

if (!paused) createImageLayer();

}

We also add the possibility to stop the stream in case we click on the picture:

function imageOnclick() { // Clicking on the image will pause the stream

paused = !paused;

if (!paused) createImageLayer();

}

We also set the required function on the <body> tag, to load the stream when we load the HTML page:

<body onload="createImageLayer();">

For the interface itself, we first define an indicator for the current status of the alarm:

<div class='row voffset50'>

<div class='col-md-4'></div>

<div class='col-md-4 text-center'>

Alarm is <span id='alarm-status'>OFF</span>

</div>

<div class='col-md-4'></div>

</div>

In the following section, we create a button to deactivate the alarm if it has been triggered:

<div class='row'>

<div class='col-md-4'></div>

<div class='col-md-4'>

<button id='off' class='btn btn-block btn-danger'>Deactivate Alarm</button>

</div>

<div class='col-md-4'></div>

</div>

Finally, we create the element that will hold the live video stream:

<div class='row voffset50'>

<div class='col-md-3'></div>

<div class='col-md-7'>

<div id="webcam">

<noscript>

<img src="http://192.168.0.105:8080/?action=snapshot" />

</noscript>

</div>

</div>

</div>

Let's now have a look at the JavaScript file. We will first link the button to the correct action on the server:

$( "#off" ).click(function() {

// Deactivate alarm

$.get('/off');

});

For the indicator of the current state of the alarm, we refresh the element of the interface every two seconds:

setInterval(function () {

// Current

$.get('/alarm', function(data) {

if (data.alarm == true) {

$( "#alarm-status" ).text("ON");

}

else {

$( "#alarm-status" ).text("OFF");

}

});

}, 2000);

It's now finally time to test the last part of the chapter! Grab all the code from the GitHub repository of the book and inside the folder where the code files are type:

**sudo npm install express request**

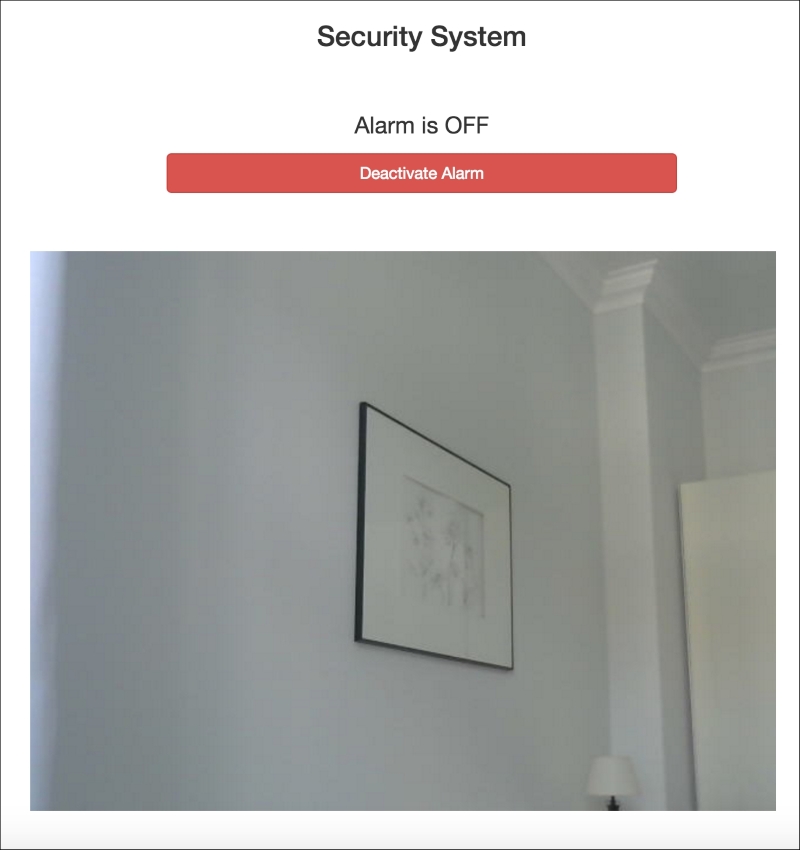
Then, start the application with the following command:

**sudo node system\_interface.js**

You can now simply navigate to the IP address of the computer or Pi on which the application is running, followed by port 3000. For example:

**http://192.168.0.100:3000**

You should immediately see the simple interface that we just created, as well as the live stream from security camera:



You can now try to pass your hand again in front of the motion sensor; you should instantly hear the sound from the buzzer and see the confirmation on your screen inside the interface. Then, simply click on the button to deactivate the alarm.

If you can't see the page, check your firewall: it might be blocking the IP address of your Pi or the port on which the application is running (3000).

**Chapter 8. Monitor Your Home from the Cloud**

In this chapter, we are going to delve more into a very exciting topic related to building a smart home: the Internet of Things. Indeed, today most of the smart homes are connected to the Internet and allows the user the monitor her or his home, even when they are at the other end of the globe.

In this chapter, we are going to learn how to build the three projects that will allow you to monitor your home from a distance. First, we are simply going to add a sensor to our Raspberry Pi Zero and monitor the measurements from a cloud dashboard. After that, we are going to learn how to build our own cloud dashboard to monitor several sensors remotely. Finally, we'll learn how to monitor the live camera stream via a wireless security camera from anywhere in the world. Let's dive in!

**Hardware and software requirements**

As always, we are going to start with the list of required hardware and software components for the project.

For the sensors, we'll use a simple DHT11 sensor, along with a 4.7k Ohm resistor. We'll also use a PIR motion sensor.

For the camera module, I will use a Logitech C270 webcam. Here, any camera compatible with the UVC protocol would work, which is the case for most of the cameras sold those days.

Finally, you will need the usual breadboard and jumper wires.

This is the list of components that you will need for this whole chapter, not including the Raspberry Pi Zero:

* PIR motion sensor (<https://www.sparkfun.com/products/13285>)
* DHT11 sensor with 4.7k Ohm resistor (<https://www.adafruit.com/products/386>)
* PIR motion sensor (<https://www.adafruit.com/products/189>)
* Logitech C270 USB camera (<http://www.logitech.com/en-us/product/hd-webcam-c270>)
* Breadboard (<https://www.adafruit.com/products/64>)
* Jumper wires (<https://www.adafruit.com/products/1957>)

To connect the camera to your Pi, I also recommend using a USB hub for this chapter, as there is only one USB port on the Pi.

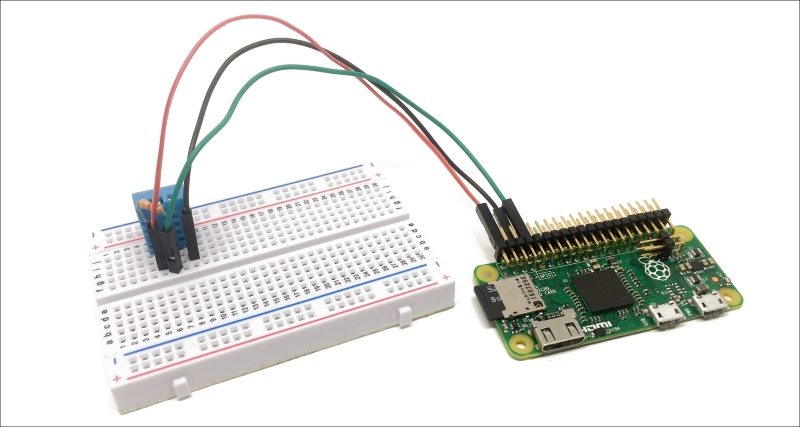
On the software side, you will need to have Node.js installed on your Raspberry Pi Zero board.

# Monitoring data from a cloud dashboard

In this first section of the chapter, we are going to connect a temperature and humidity sensor to our Raspberry Pi Zero board and send those measurements to the cloud. Later in this section, we are also going to learn how to visualize those measurements on a dashboard.

We first need to connect the DHT11 sensor to our Pi. First, place the sensor on the board, and then connect the 4.7k Ohm resistor between pin 1 and 2 of the sensor. Then, connect the first pin of the sensor to a 3.3V pin of the Pi, the second pin to GPIO4 of the Raspberry Pi, and finally the last pin of the sensor to a GND pin of the Pi.

The following image is the final result:



We are now going to see how to configure our Raspberry Pi Zero so it automatically sends data to the cloud. For that, we'll use Node.js to send data to a service called **Dweet.io**, which will allow us to easily store data online.

Let's first see the details of the code. First, we declare the modules that we will use for this section:

var sensorLib = require('node-dht-sensor');

var request = require('request');

After that, we need to give a name to our thing, which is the name we'll use to identify the object storing the measurements on Dweet.io:

var thingName = 'mypizero';

We will also define a main measurement loop, in which we'll make measurements from the sensor and send those measurements to Dweet.io:

var sensor = {

initialize: function () {

return sensorLib.initialize(11, 4);

},

read: function () {

// Readout

var readout = sensorLib.read();

console.log('Temperature: ' + readout.temperature.toFixed(2) + 'C, ' +

'humidity: ' + readout.humidity.toFixed(2) + '%');

// Log data

logData(readout);

// Repeat

setTimeout(function () {

sensor.read();

}, 2000);

}

};

After that, we need to initialize the sensor:

if (sensor.initialize()) {

sensor.read();

} else {

console.warn('Failed to initialize sensor');

}

Let's now see the details of the function that is used to log data on the Dweet.io server:

function logData(readout) {

// Build URL

var url = "https://dweet.io/dweet/for/" + thingName;

url += "?temperature=" + readout.temperature.toFixed(2);

url += "&humidity=" + readout.humidity.toFixed(2);

// Make request

request(url, function (error, response, body) {

if (!error && response.statusCode == 200) {

console.log(body) // Show response

}

});

}

We basically form a request to Dweet.io, passing the measurements inside the request URL itself.

It's finally the time to test the project! Grab all the code from the GitHub repository of the book and place it inside a folder on your Pi. Then, inside this folder, type the following command with a terminal:

**npm install node-dht-sensor**

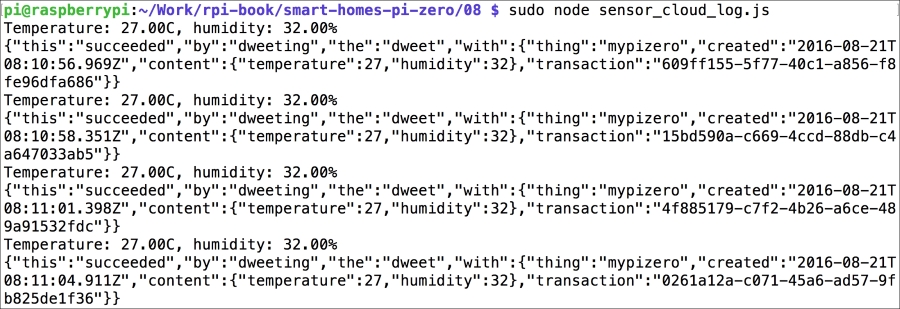
This will install the required sensor library. Then, install the requestmodule with the following command:

**sudo npm install request**

Finally, you can start the software by typing:

**sudo node sensor\_cloud\_log.js**

You should immediately see the answer from Dweet.io as the data is recorded to the cloud:



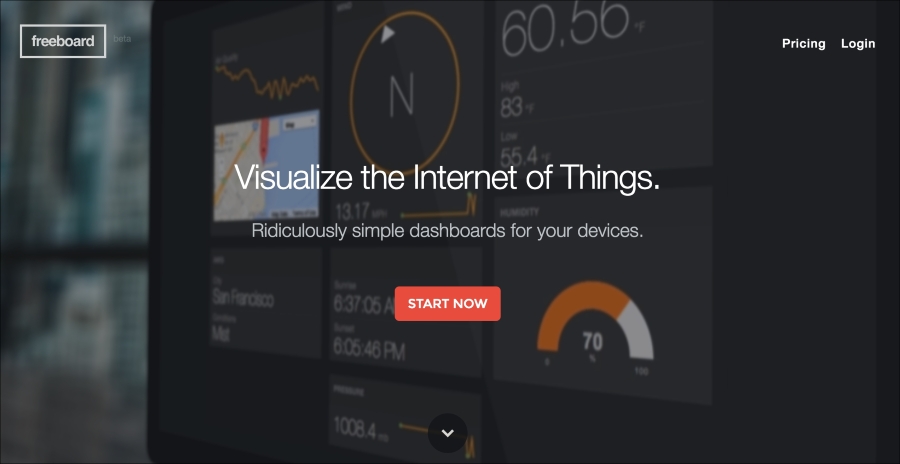
You can actually already visualize this data right in your web browser, by typing the following URL:



This is nice, but it's not great to actually visualize data as it is recorded. That's why we are now going to use Freeboard.io, which is a service that will allow us to create cloud dashboards using the Dweet.io data.

You can already create an account at:

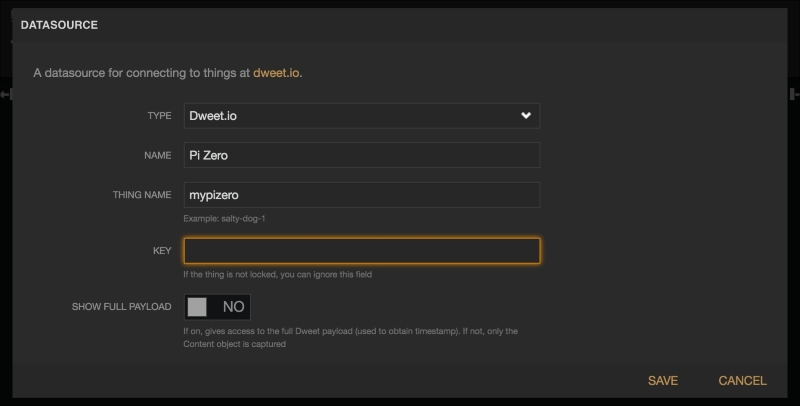
<http://freeboard.io/>



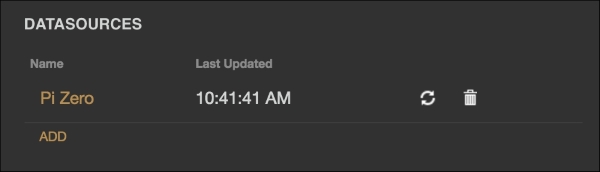
Inside Freeboard.io, first create a new dashboard:



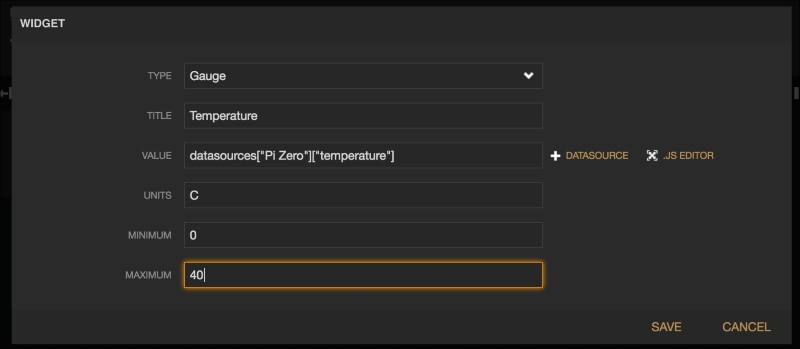
Then, add a new datasource with the following parameters:



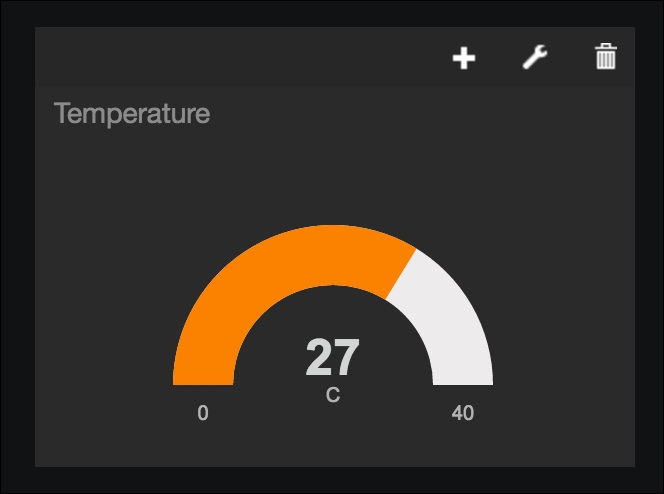
This will basically link your dashboard to the 'thing' that is storing your data on Dweet.io. After that, you'll see that the connection is active inside the dashboard itself:



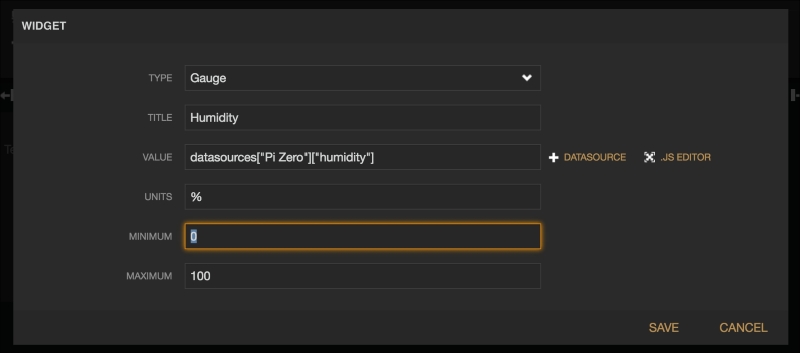
Now, create a new pane inside your dashboard and also a new **Gauge** widget for the temperature, using the following parameters:



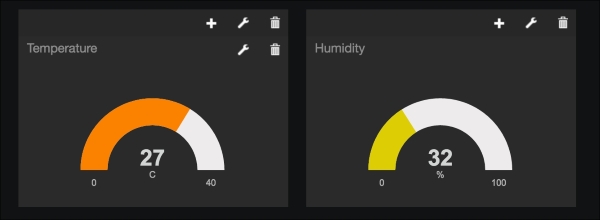
You should be able to immediately see the temperature measurements being displayed in the dashboard:



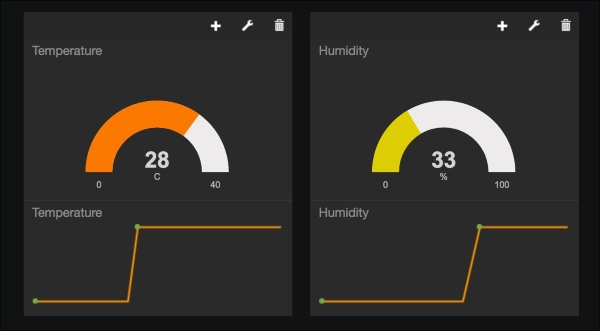
Now, do the same for humidity, using the following parameters:



You should now have both gauges inside your dashboard, giving you an immediate glance at the temperature and humidity inside your home:



You can now add more visualizations of the same data. For example, I added two additional widgets of the type **sparkline** for each measurement, giving me an instant view of the recent history of each variable:

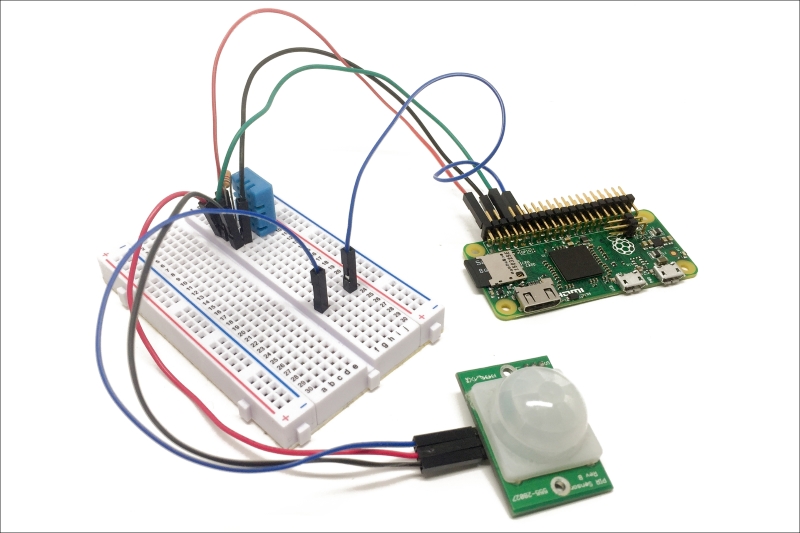


# Creating a cloud dashboard for your devices

In the second part of this chapter, we are going to add a motion sensor to the project we built in the first part and also learn how to monitor all those sensors from a single dashboard. I will connect all the sensors to a single Raspberry Pi Zero board, but you could of course have them connected to several boards that are in different parts of your smart home.

The project itself will be really easy to assemble. First, make sure that you followed all the instructions from the previous project. Then, simply connect the motion sensor to the project: VCC goes to the 3.3V pin of the Raspberry Pi, GND to GND, and the SIG pin of the sensor is connected to Raspberry Pi GPIO18.

The following image is the final result:



Let's now see how to configure the project. In order to access the measurements from anywhere in the world, we'll use the aREST framework again, which we have already used in several projects of the book. However, here we'll use the cloud access of aREST that will allow us to access those measurements from anywhere.

Inside the code itself, we first include all the required modules:

var sensorLib = require('node-dht-sensor');

var express = require('express');

var app = express();

var piREST = require('pi-arest')(app);

We then define the ID and the name of the board:

piREST.set\_id('73gutg');

piREST.set\_name('pi\_zero\_cloud');

piREST.set\_mode('bcm');

Note that as the ID is unique for each Raspberry Pi board, you need to change it and insert your own ID here. Then, inside the main measurement loop, we expose the temperature and humidity measurements to the aREST framework:

piREST.variable('temperature', readout.temperature.toFixed(2));

piREST.variable('humidity', readout.humidity.toFixed(2));

We also do the same for a variable called motion that depends on the current state of the motion sensor:

piREST.digitalRead(18, function(data) {

if (data == 1) {

piREST.variable('motion', "Motion Detected");

}

else {

piREST.variable('motion', "No Motion");

}

});

After that, we connect to the aREST cloud server:

piREST.connect();

Finally, we start the server with the following code:

var server = app.listen(80, function() {

console.log('Listening on port %d', server.address().port);

});

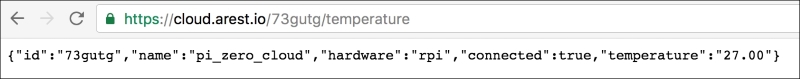
It's now finally time to test the project! Inside the same folder as you put the files of the first project of this chapter, type:

**sudo npm install express pi-arest**

When the modules are installed, start the project using:

**sudo node sensor\_cloud\_arest.js**

This will immediately make the project connect to the aREST.io cloud server. You can actually test it by typing the following URL inside your favorite web browser, of course by changing the ID of your device:



In order to display this data inside a dashboard, we are going to use the dashboard of the aREST framework that you can access from:

<http://dashboard.arest.io/>

From there, create a new account and then a new dashboard:

Creating a cloud dashboard for your devices

Inside this dashboard, add a new element with the following parameters, which will be used to display the temperature inside the dashboard:

Creating a cloud dashboard for your devices

Once it is done, you can do the same for humidity. You should now have both the data showing up inside your dashboard:



Finally, do a similar operation with the motion variable:

Creating a cloud dashboard for your devices

You should now have all the variables measured by your Pi displayed inside the same dashboard:

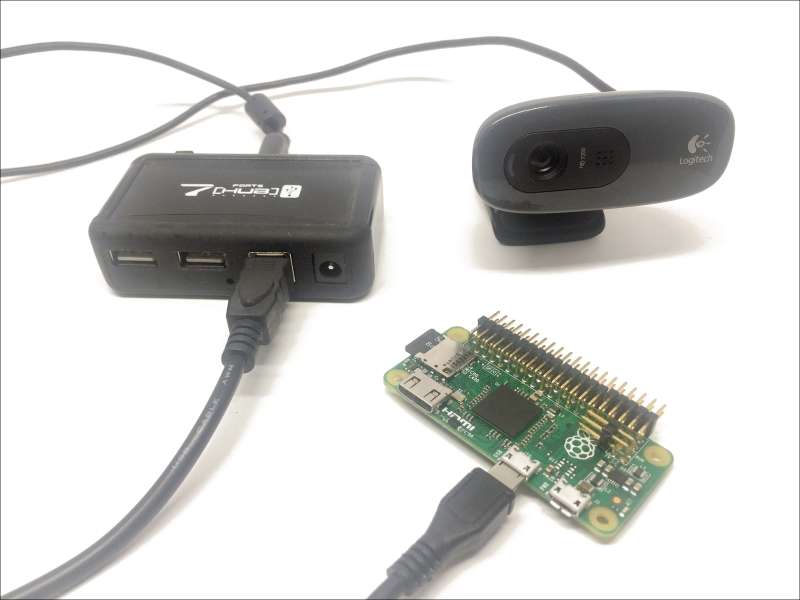


Congratulations, you can now monitor your home from anywhere in the world! Also note that you can have measurements from several Raspberry Pi boards displayed in the same cloud dashboard.

# Accessing your security camera from anywhere

Inside the last section of this chapter, we are going to revisit a project we built in the last chapter: the wireless security camera. Here, we are going to learn how to visualize the live video stream coming from a camera from anywhere in the world.

Assembling this project is really simple; you simply need to plug the USB camera to the Raspberry Pi using the USB hub as you also need to plug the Wi-Fi dongle. This is the final result:

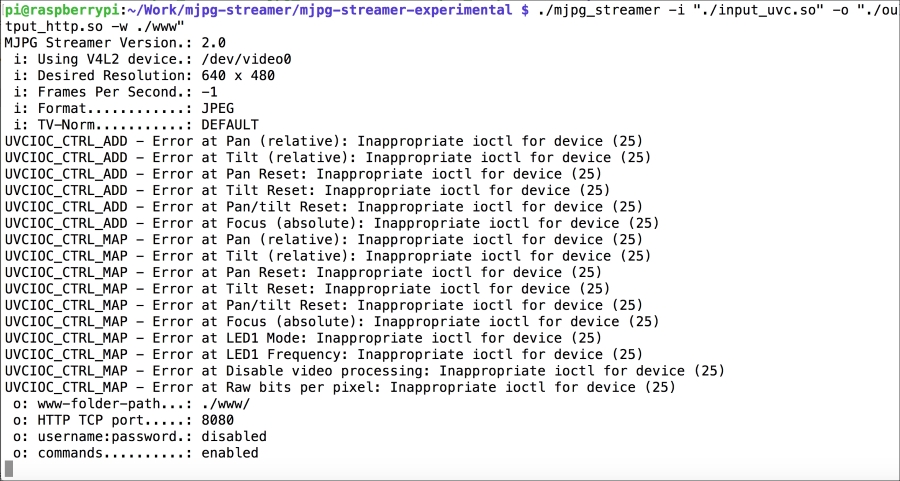


Now, we are again going to use the MJPEG-streamer software to create a live video stream from the camera. If you haven't installed it yet, you can check the last chapter of the book to learn how to install it.

Then, go inside the folder where the software is installed, and type:

**./mjpg\_streamer -i "./input\_uvc.so" -o "./output\_http.so -w ./www"**

This will immediately start the streaming software on your Raspberry Pi:



Now, I want you to open another terminal window or tab, as we will need to run another software while the streaming software is running. We are going to use software called **Ngrok**; this will allow us to access the video stream from anywhere in the world.

In the second terminal window, type:

**wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-arm.zip**

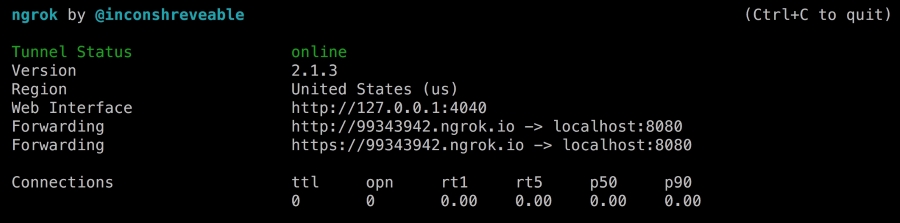
This will download Ngrok on your computer. Then, unzip the file with:

**unzip ngrok-stable-linux-arm.zip**

Finally, start Ngrok using the following command:

**./ngrok 8080**

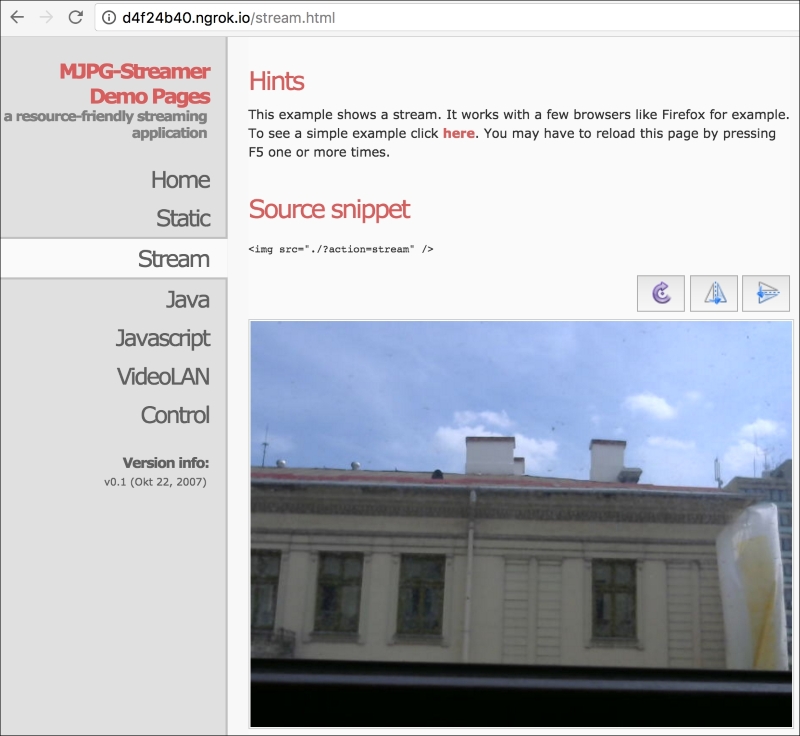
This will basically create a web URL that you can use to access your Pi on port 8080, which is precisely the port on which the streaming software is running. Once Ngrok is running, you should be able to see the URL you need inside a window:



You can now simply copy this URL and type it inside any web browser. You should then immediately be able to see the interface created by the streaming software:



Now, simply click on **Stream** to see the live stream coming from the board:



Congratulations, you can now access your wireless security camera from anywhere in the world and monitor your home remotely!