Build a cloud-ready temperature sensor with the Arduino Uno and the IBM Watson IoT Platform, Part 1: Build the circuit and set up the environment

Kyle Brown

September 16, 2014

This tutorial introduces you to the Arduino microcontroller, discusses how the author has used it to solve a home-control issue, introduces some hardware add-ons to the Arduino and shows how to connect them up, and then shows how to load programs to test the hardware.

View more content in this series

Important: IBM Internet of Things Foundation (IoT Foundation) is now named IBM Watson IoT Platform. The content and images have not been updated. It is provided "as is." Given the rapid evolution of technology, some steps and illustrations may have changed as well.

Many of you might live in a technology-saturated home as I do. At the center of this web of connected devices sits my home WiFi router, connected to the Internet through a cable modem. So, when all of a sudden you hear cries from throughout the house because laptops, tablets, game boxes, DVRs, and all the rest of the Internet-connected devices have failed because the WiFi router has stopped working, you know something is amiss.

In the past, I've tried multiple approaches to improve this situation: I've replaced the router with a new model, I've updated the firmware, I've replaced the cable modem, and I've had the cable company check the signal on the coax lines — all to no avail. The consensus in the Internet community among those with similar problems is that either the router or the modem (or both) might be getting too hot. That seemed unlikely to me because it doesn't feel warm in the wiring closet where I installed these machines, but my evaluation of the temperatures has not been scientific sampling. What I needed was a way to find out the temperature of the wiring closet when or just before the failures occurred so I could see if there was a correlation. And that need led me to my experiment.

Like many other engineers, I've been entranced for a number of years by the capabilities of the Arduino open source electronics platform. The Arduino is a single-board microcontroller that was introduced in 2005 (and named by its creators after a bar in the Northern Italian town of Ivrea).
The Arduino is programmed by using a language (called Processing) that's based on the Java™ language. The Arduino community is enormous, and hundreds of examples of using the board to take measurements from various sensors and control all sorts of actuators exist on the web. So when I thought about my problem, my attention turned to an unused Arduino Uno board sitting beside my desk, together with the Arduino Ethernet shield sitting in my drawer and the brand-new temperature sensor that had just arrived the week before that was waiting for a project. And as I thought about the IBM Internet of Things (IoT) Foundation and the capabilities for IoT development that are part of IBM Cloud, the concept for my project began to take shape. Figure 1 shows the entire project that I wanted to build.

Figure 1. Illustration of the project's conceptual design

In the design, the Arduino board, connected to a temperature sensor, sends temperature (Fahrenheit and Celsius) and humidity information on a regular schedule to the IoT Foundation via the MQTT protocol and is graphed, using capabilities in IBM Cloud. The IoT Foundation enables this data to be plotted on a realtime graph. You'll complete this entire project over the course of this series.

“As I thought about the IBM Internet of Things Foundation and the capabilities for IoT development that are part of IBM Cloud, the concept for my project began to take shape.”

Getting started with an Arduino

If you want to follow along and build the same sensor that I built, you first need any of the many Arduino boards or Arduino clones that are out there. (But be careful — although many boards, such as the Intel Galileo, claim to be "Arduino compatible," there are enough differences in libraries and tolerances of the internal electronics to make using a "compatible" board sometimes tricky.) I'm fond of the Arduino Uno R3, which is the newest revision of the "classic" Arduino based on the ATMega128 chipset. You can easily find this board in brick-and-mortar stores like Radio Shack, and from Amazon and other online retailers, for under US$30. Figure 2 is a photo of the top side of the Uno R3.
When you first receive your board, visit the Arduino website, which is the home of the Arduino community and contains a wealth of information and documentation on the boards, their capabilities, and the software that has been written to take advantage of them. For this quick "cook's tour," I'll simply start at the top of the board and walk you around the outer edge to review the most important parts of the board from the perspective of this tutorial.

At the very top sits a row of holes for pins that can be used to read values from, or to control digital devices. Moving to the bottom is a row that is half-occupied by pins for reading or writing analog values, while the remaining pins on that row provide different types of power and ground connections. On the bottom left of the board is a jack for a 5V power supply; you'll use this jack if you want to power your project when it's not connected to your computer. The next jack above that on the left side of the board is a connector for a USB cable. This connector is used both for communication to the computer you program the board from, and for powering the board when the cable is connected. Finally, at the top left of the board is the reset button — just the thing for when your software is misbehaving.

One of the strengths of the Arduino community is the fact that not only has the basic board had its specifications released into the open source community, resulting in many versions of it, but also that many different shields have been built for it. The one for this project is the R3 version of the Ethernet shield, which provides access to an Ethernet connection at either 10 or 100MB through a standard RJ45 connector. It's also available from many of the same retailers as the Arduino Uno, for about US$30. Figure 3 shows the Ethernet shield.
The neat thing about these boards is that one plugs into another — so a project might have an Arduino on the bottom, an Ethernet or WiFi shield in the middle, and perhaps a motor shield (for Robot control) or an additional extension shield for special sensors on the top. You can get a sense of what the boards look like when stacked from the photo in Figure 4.

Once you've stacked up the boards, you might feel like you're ready to go, but wait: You must first download and install the Arduino IDE, which is available from the download link at the Arduino website and is compatible with Mac, Windows®, and Linux®. If you want to, you can try one of the...
newer beta versions, but for this tutorial I'll use the current released version of the IDE, which is Arduino 1.0.5.

Download the IDE (which usually comes as a ZIP or TAR file, depending on your computer's OS) and follow the instructions to install the IDE and the USB drivers on your computer. At that point, you can then plug in the USB cable to power up the board (and often complete the driver installation), plug in your Ethernet cable, open the IDE, and start trying out the library examples that come with the IDE — but that's getting a bit ahead of the example.

**Building the circuit**

For this project's purposes, you have a few more pieces of hardware to acquire. The first is a standard prototyping "breadboard." If you're an electrical engineer or electronics hobbyist, you're undoubtedly familiar with these already. If you're not, then you'll need to purchase one like the one shown in Figure 5.

**Figure 5. Prototyping breadboard**

The same retailers I mentioned earlier carry these for about US$5 by themselves, or perhaps as much as US$10-15 when they're included with a set of jumper wires (which is another necessary set of hardware). For the uninitiated: The way a breadboard works is that the numbered rows on each side are all are electrically connected together, so pins a through e in row 1 are all connected together on either side. If you insert the lead wire of an electrical component such as a resistor into pin e in row 1, you can plug a wire into pin d in row 1 that could then connect to another
component in another row. Using the breadboard, you can quickly prototype electronic circuits temporarily — a fast and easy way to try out connections between different components.

So what are you going to wire together? Well, the Arduino and the Ethernet shield have the ability to read digital measurements (1s or 0s) from any of the digital pins along the top. You can then write programs in Processing to send those readings (or the interpretation of those readings) to other computers. The specific thing you want is a temperature reading — and for that, you need a temperature sensor.

Several Arduino-compatible temperature sensors are available, but the one I used for my project is the Virtuabotix DHT11 temperature and humidity sensor. It's available from either the Virtuabotix website or Amazon for less than US$10. This tiny four-pin wonder is shown in Figure 6.

**Figure 6. DHT11 temperature and humidity sensor**

![DHT11 temperature and humidity sensor](image)

When you receive the DHT11 sensor, it should come with a set of pinout documentation showing what all the pins on the sensor do. If not, you can find that information at the Virtuabotix website.

While you are visiting the Virtuabotix website, download the Arduino libraries (which come packaged as a ZIP file) for the Virtuabotix DHT11. They are available from the download link on that same page. Extract them into the Libraries subdirectory of the Arduino IDE directory structure. When you restart the Arduino IDE, you'll see that the DHT11 samples are available for use, just like the built-in samples for the Ethernet shield and other devices.

Now that you have all the physical pieces, it's time to wire together your project. The wiring diagram in Figure 7 shows all the connections that you need to make.
Let’s review the wiring of these parts. The Arduino is placed on the bottom, while the Ethernet shield is placed on top of it (see Figure 4 for an example of how this works). I simply plugged the DHT11 into an unused column of pins on the breadboard. Note that the front of the DHT11 with the waffle grille — not the side with the label — is facing forward. The ground pin on the DHT11 (which is the one on the far right side with the waffle grille facing forward) is connected via the black wire to the GND pin on the Arduino. The VDD pin on the DHT11 (the one on the farthest left) is connected via the red wire to the 5V power-supply pin on the Arduino. Finally, the DTA pin (second from the left) is connected to Digital pin 3 on the Arduino. Remember that pin number —pin 3— because that becomes important when you write the software that references the DHT11.

Your final configuration should look something like Figure 8.
At this point, it's a good idea to individually test out the components. You might want to start by testing for the proper installation of the USB drivers in your computer to see if it is successfully communicating with the Arduino. Open the Arduino IDE and click **File > Examples > 01.Basics > Blink**, as shown in Figure 9.

The **Blink** sketch (programs in the Arduino are called *sketches*) is a simple one that enables you to test communication to the Arduino. All it does is cause one of the built-in LEDs on the Arduino board to blink on and off with a period of one second. If you correctly installed the USB device drivers, you should be able to select your board type by clicking **Tools > Board** (for instance, if you're using a Uno, click that) and your serial Port from the SerialPort menu (refer to your OS tools such as the Windows Device Manager for the name or number of the serial port you are using). If either of these selections is unavailable, refer to the help on the Arduino website. After you select your board and your serial port, you can upload the sketch to the board by clicking the **Upload** button (the right-pointing arrow, second button from the left in the toolbar), as shown in Figure 10.
If the message **Done uploading** is displayed and you see the LED begin to blink, then you're successfully in communication with the Arduino. If not, again, refer to the [Arduino website](https://www.arduino.cc) for help. Once you've proven that you can connect to the Arduino, you should try one or more of the Ethernet shield examples, by clicking **Examples > Ethernet > Web Server**.

A trick for making the web server example functional is that it uses two parameters that you probably need to change. Near the top of the example sketch, you'll see the following lines of code:

```cpp
// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
IPAddress ip(192,168,1,177);
```

On the newer Ethernet shields, the MAC address is on a sticker attached to the shield. (For older boards, you'll need to make one up.) Change the `mac[]` variable values in the code to match the MAC address shown on the sticker. Likewise, you need to change the IP address to an open address on your local network. Make these changes, save the sketch to a new name, and then upload it. It should work now. Finally, if you downloaded and extracted the DHT11 libraries to the correct directory, you should be able to click **Examples > DHT11 > dht11_functions** and load that sketch. Near the top of that sketch, you'll see the following code:

```cpp
void setup()
{
  DHT11.attach(2);
  Serial.begin(9600);
  Serial.println("DHT11 TEST PROGRAM ");
  Serial.print("LIBRARY VERSION: ");
  Serial.println(DHT11LIB_VERSION);
}
```

Remember that you attached the DHT11 DTA pin to pin 3 on the Arduino? That's because pin 2 is used by the Ethernet shield and is unavailable. So change the 2 in the line `DHT11.attach(2)` to a 3, save the sketch to a different name, and then upload it. Then open the Serial Monitor (click **Tools > Serial Monitor**) and you should see output from the program showing temperature and humidity data scrolling down the screen.
Conclusion and next steps

Now that you've proven that all of the parts are functional, you're ready to move on to the next part of the series. In Part 2, I explain the communications protocol of the IoT — MQTT — and discuss the Arduino client libraries for MQTT as well as some hints and techniques for testing your MQTT programs locally. I also discuss the capabilities of the IBM IoT Foundation, show you how to connect a program running on the Arduino into the IoT Foundation, and bring all the pieces together for showing how the sensor data comes in, is formatted for the IoT, and is sent to the cloud.
Related topics

- **IBM IoT Foundation**: Try out the IBM IoT Foundation and sign up for the beta program.
- **Arduino**: Visit the Arduino website.
- **DHT11**: Learn more about the DHT11 sensor from the Virtuabotix website.
- **Arduino IDE**: Download the IDE from the Arduino website.
- **Libraries for the DHT11 Temperature Sensor**: Download the DHT11 libraries.

© Copyright IBM Corporation 2014

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)