Build your own dashboard to control your devices from the cloud

We are now able to control any board from the cloud using the aREST cloud API. In this chapter, we are going to see how to create our first online dashboard to control an ESP8266 board from the cloud, using aREST.

You will be able to build your own dashboard to control a LED remotely. We'll also see how to read data from your board and display it inside the dashboard. Let's start!

Hardware & Software Requirements

The hardware requirements for this chapter are really similar to other chapters in which we already used the ESP8266 chip.

This is the list of the required hardware for this chapter:

- ESP8266 Huzzah module (https://www.adafruit.com/products/2471)
- FTDI USB converter (https://www.adafruit.com/products/70)
- LED (https://www.sparkfun.com/products/9590)
- 330 Ohm resistor (https://www.sparkfun.com/products/8377)
- Breadboard (https://www.sparkfun.com/products/12002)
- Jumper wires (https://www.sparkfun.com/products/12795)

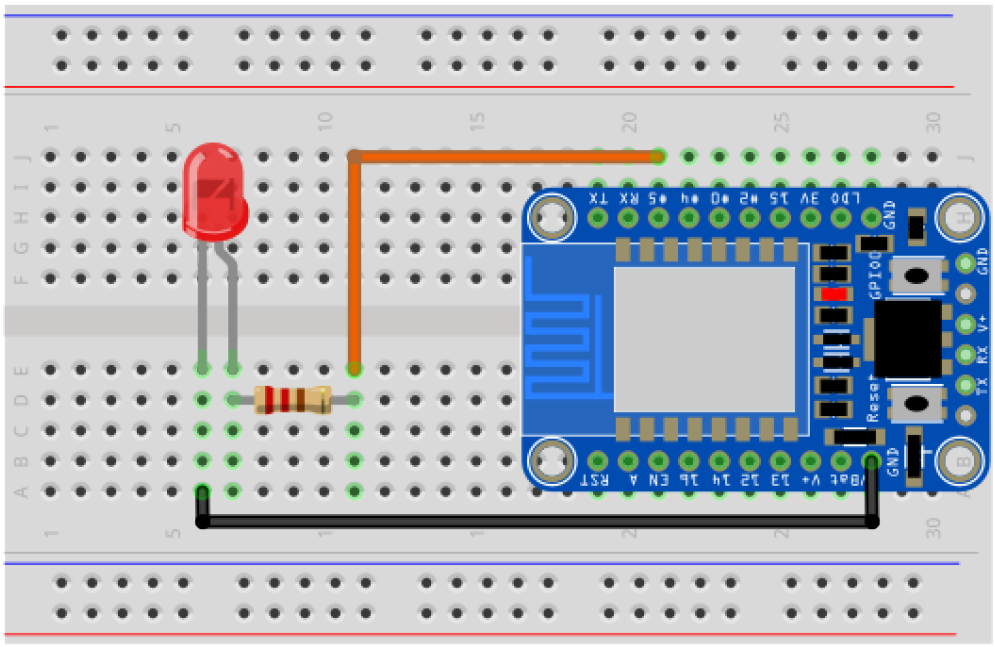
For the software part, you will need to have the latest version of the Arduino IDE installed, as well as the following libraries:

- aREST
- PubSubClient

You also need to have the ESP8266 board definitions installed inside your Arduino IDE.

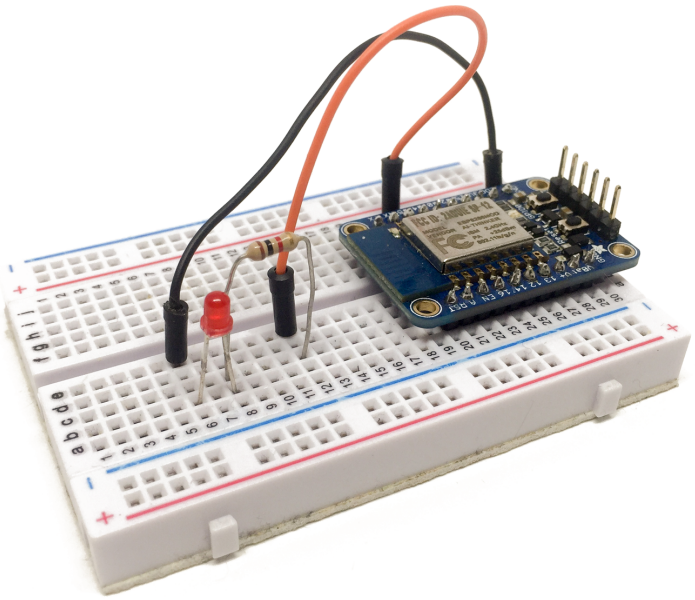
Hardware Configuration

Let's now assemble the hardware for this chapter. To help you out, this is the schematic for this chapter:



First, place the ESP8266 board on the breadboard. Then, place the resistor in series with the LED. Connect the resistor to pin number 5 of the ESP8266 board, and the other end of the LED to one GND pin of the ESP8266.

This is the final result:



Setting Up Your ESP8266 Board

It's now time to configure the ESP8266 board. This is the complete code for this chapter:

```
// Import required libraries
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <aREST.h>

// Clients
WiFiClient espClient;
PubSubClient client(espClient);

// Create aREST instance
aREST rest = aREST(client);

// Unique ID to identify the device for cloud.arest.io
```

```
char* device_id = "3g83df";

// WiFi parameters
const char* ssid = "your-wifi-name";
const char* password = "your-wifi-password";

// Variables to be exposed to the API
int temperature;
int humidity;

// Functions
void callback(char* topic, byte* payload, unsigned int length);

void setup(void)
{
  // Start Serial
  Serial.begin(115200);

  // Set callback
  client.setCallback(callback);

  // Init variables and expose them to REST API
  temperature = 24;
  humidity = 40;
  rest.variable("temperature",&temperature);
  rest.variable("humidity",&humidity);

  // Give name and ID to device
  rest.set_id(device_id);
  rest.set_name("esp8266");

  // Connect to WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
```

```
  Serial.println("WiFi connected");

  // Set output topic
  char* out_topic = rest.get_topic();

}

void loop() {

  // Connect to the cloud
  rest.loop(client);

}

// Handles message arrived on subscribed topic(s)
void callback(char* topic, byte* payload, unsigned int length) {

  rest.handle_callback(client, topic, payload, length);

}
```

For this chapter, I didn't want to complicate things by using a sensor, so I just defined two variables that contains data:

```
int temperature;
int humidity;
```

In the setup() function, I assigned some arbitrary values to those variables:

```
temperature = 24;
humidity = 40;
rest.variable("temperature",&temperature);
rest.variable("humidity",&humidity);
```

Of course, you could perfectly connect a sensor to your board, and use the readings from this sensor to feed the aREST variables that we declared in the sketch.

As usual, you will also need to change the ID of the device in the sketch, as well as you WiFi name & password.

After that, put the board in bootloader mode, and upload the code to the board. You can quickly check in the Serial monitor that the board is indeed connected to aREST.io.

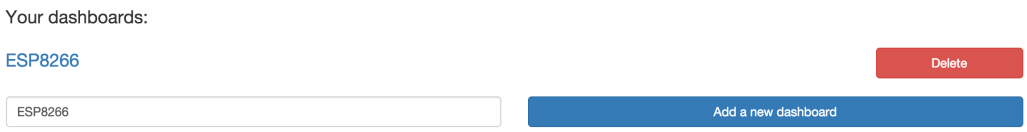
Controlling Output

Let's now see how to control the LED from a dashboard that we will create. To do that, we need to introduce a new tool inside the aREST framework: the cloud dashboard service, which is available at:

http://dashboard.arest.io/

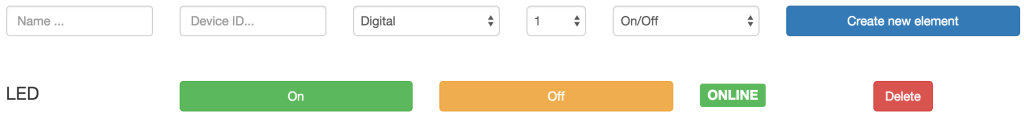
Go to this URL, and create an account. You will instantly be taken to the page where you can create a new dashboard:

After creating a new dashboard, you will see it in the main window:



Click on your newly created dashboard, and create a first element to control the LED on the board. Put a name, the ID of your device, and also assign pin 5 to this dashboard element. This is how it looks like:

Then, add the element by clicking on the **Create** button, and you will see that the element has been added to the dashboard:



Finally, it's time to test your dashboard element! Simply click on the **ON** button: you should immediately see the LED turning on on your board.

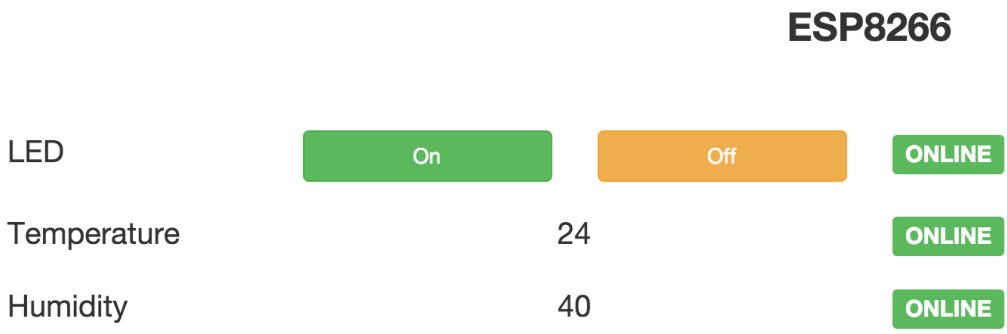
Also note that every time you create a new dashboard element to control a digital pin, this pin is automatically set as an output by the cloud API.

Displaying Data

We are now going to move further with the dashboard we created earlier, and display the data that is present on the board inside the dashboard.

Go back to the dashboard, and create a new element with the same device ID, of the *variable* type, linked to the temperature variable:

You will rapidly see the value of the variable displayed in your dashboard. You can now do the same for the temperature variable:



How to Go Further

You can now create a dashboard to control a single board, in the cloud, using the aREST framework. You can of course use what you learned in this chapter to control all kind of on/off devices from your dashboard, for example a relay.

You can also use the cloud dashboard service to display temperature, humidity, or measurements coming from other sensors.