Lab 1: connect a virtual sensor and a Raspberry Pi device to a Node-RED application

Overview

Let’s say you are monitoring the temperature of a greenhouse. You want to learn whether the temperature exceeds 40 degrees Celsius so that you can alert your colleagues on social networks.

This lab shows you how to use Node-RED to connect the Raspberry Pi device to the IBM® Bluemix® cloud. You will use the Node-RED application to push data from the temperature sensor on the Raspberry Pi (the temperature of the CPU) to the Node-RED application that is running IBM Bluemix.

In Bluemix, you can easily add Twitter, a Cloudant database, and other Cloudant services to your application to enhance your application.

This hands-on lab shows you how to work in two Internet of Things (IoT) environments, both running Node-RED, which is a tool for wiring together hardware devices, APIs, and online services in new and interesting ways.

1. You'll create a Node-RED flow in IBM Bluemix that displays temperature values from a virtual thermometer. These values can come from a greenhouse, a data center, an airplane engine, or from many other places.
2. You'll then create a Node-RED flow on the Raspberry PI that displays the temperature of the Raspberry PI CPU.
3. Finally, you'll connect the flow on the Raspberry PI to the Node-RED flow on Bluemix so that you can see live temperature data from the Raspberry PI on the Node-RED flow on Bluemix.

Prerequisites

You need an IBM Bluemix account, experience using Node-RED, and basic knowledge of the Raspberry Pi device. For help setting up your device, see the Raspberry Pi website.

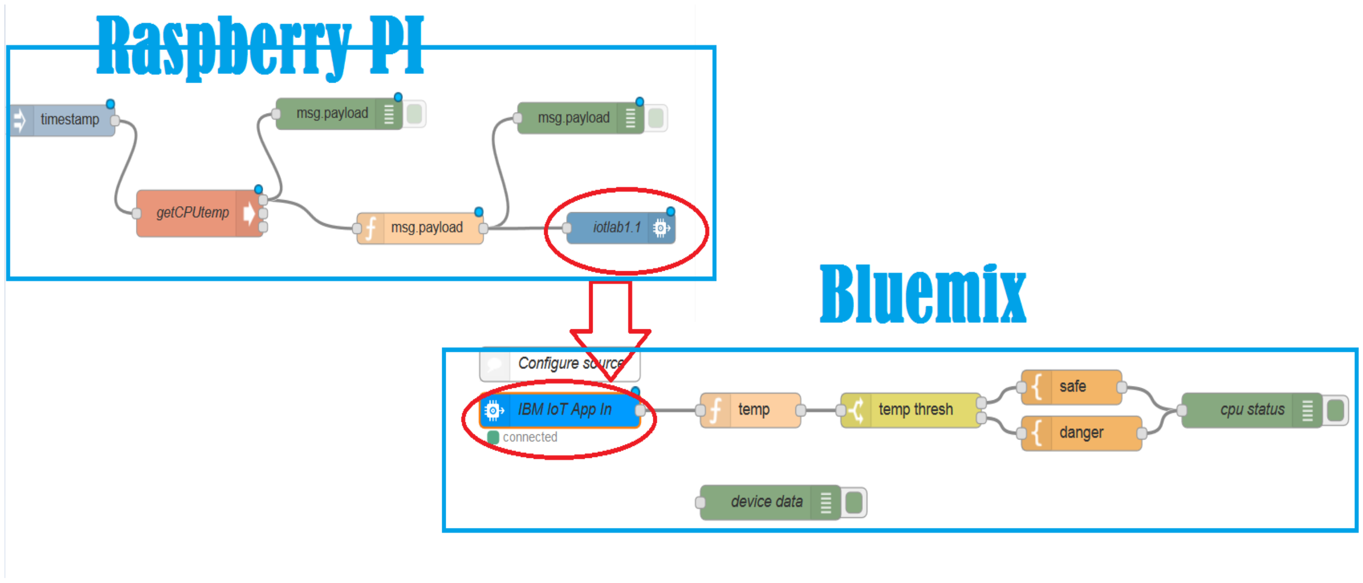
Code snippets for this lab are available on GitHub: https://github.com/blumareks/iot-watson-swift.

Application architecture

The architecture that allows the Raspberry PI to send sensor data to Bluemix has two components built with the Node-RED Internet of Things (IoT) environment.

A flow in Node-RED on the Raspberry PI captures sensor data and sends that data to a Node-RED environment on Bluemix. After the data reaches Bluemix, that data can be stored in a database and analyzed with advanced services such as the IBM Watson services.

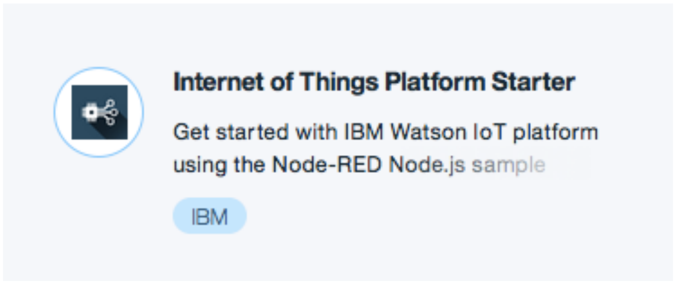
This distributed environment is common to most IoT implementations.



Step 1. Create the Internet of Things Platform Starter in IBM Bluemix and connect to a virtual sensor

In this section, you'll deploy the Internet of Things Platform Starter, which is a boilerplate application, and connect it to a virtual sensor on a web page. You don't need your Raspberry Pi device for this section.

1. Log in to Bluemix and click **Catalog > Boilerplates > Internet of Things Platform Starter**.

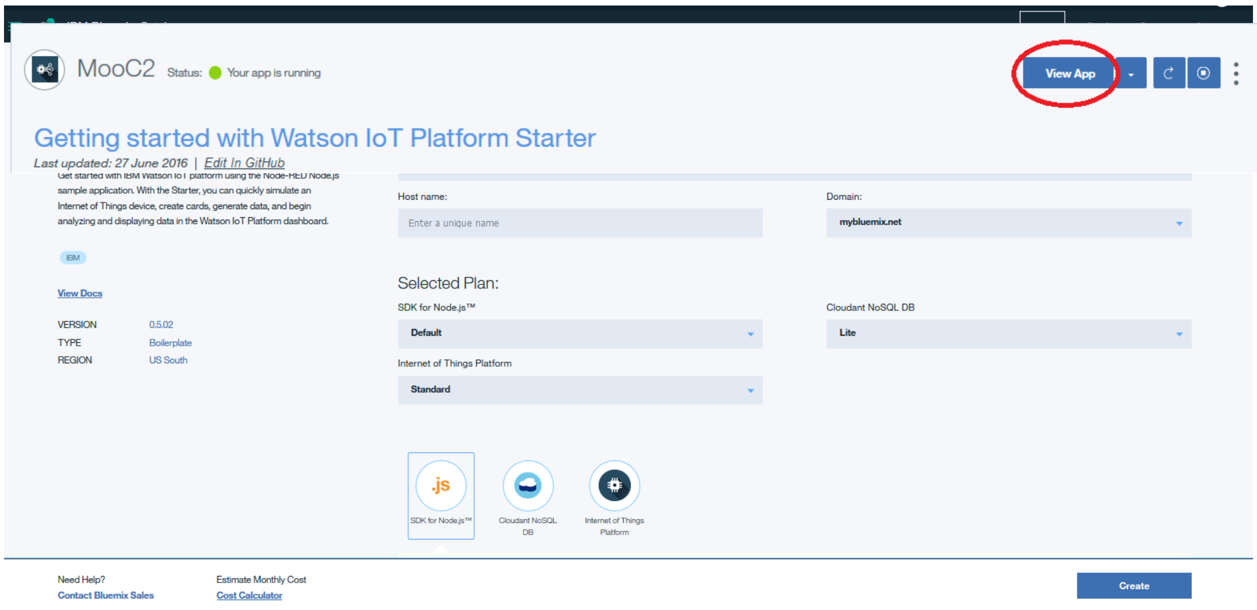


Internet of Things Platform Starter

Get started with IBM Watson IoT platform using the Node-RED Node.js sample

IBM

2. Enter a name and host for your application. Both names must be unique. Then, click **Create**.



MooC2 Status: Your app is running

Getting started with Watson IoT Platform Starter

Last updated: 27 June 2016 | Edit In GitHub

Get started with IBM Watson IoT platform using the Node-RED Node.js sample application. With the Starter, you can quickly simulate an Internet of Things device, create cards, generate data, and begin analyzing and displaying data in the Watson IoT Platform dashboard.

IBM

View Docs

VERSION	0.5.02
TYPE	Boilerplate
REGION	US South

Host name: Enter a unique name

Domain: mybluemix.net

Selected Plan:

SDK for Node.js™: Default

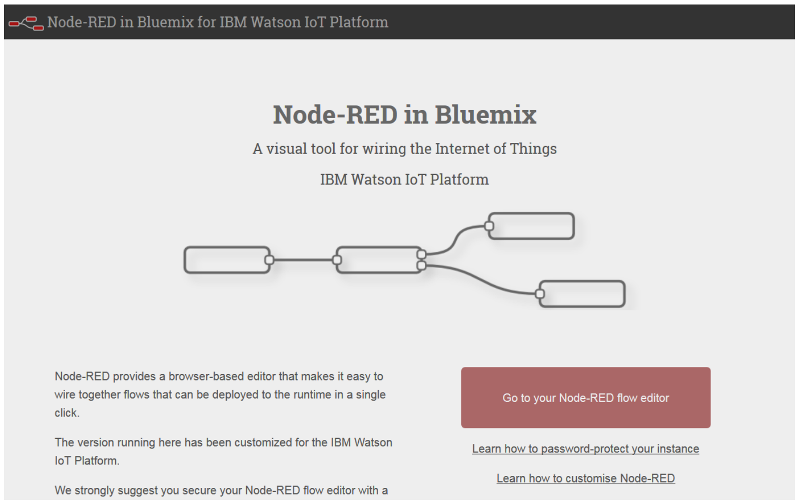
Cloudant NoSQL DB: Lite

Internet of Things Platform: Standard

Need Help? Contact Bluemix Sales

Estimate Monthly Cost Cost Calculator

3. After the application launches, click **View App** and then on go to your Node-RED flow editor.


Node-RED in Bluemix for IBM Watson IoT Platform

Node-RED in Bluemix

A visual tool for wiring the Internet of Things

IBM Watson IoT Platform

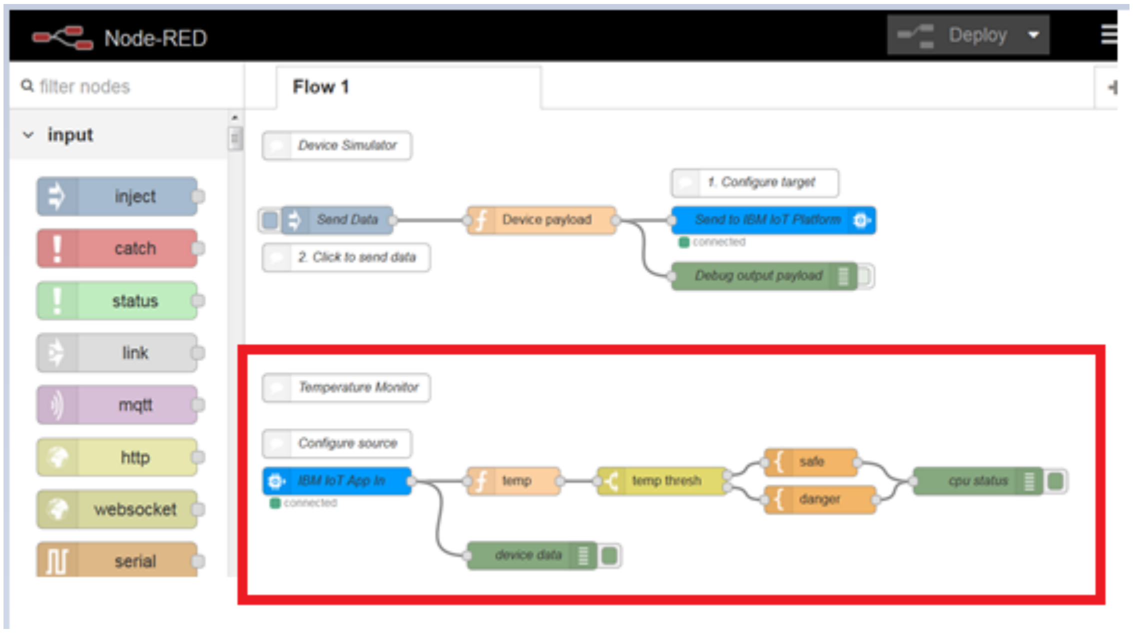
Node-RED provides a browser-based editor that makes it easy to wire together flows that can be deployed to the runtime in a single click.

The version running here has been customized for the IBM Watson IoT Platform.

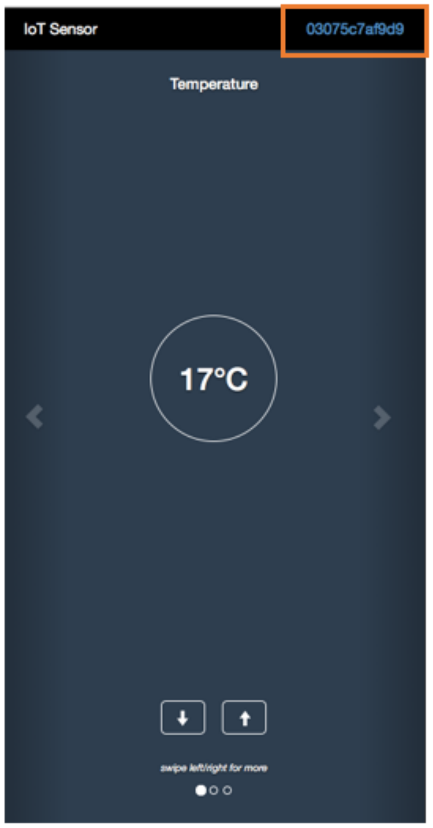
We strongly suggest you secure your Node-RED flow editor with a

Go to your Node-RED flow editor
Learn how to password-protect your instance
Learn how to customise Node-RED

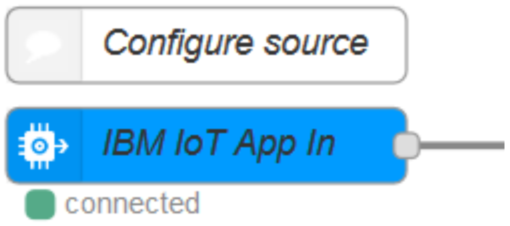
You should see Flow 1 running on Bluemix with a virtual thermometer. Ignore the flow above the flow in the red square.



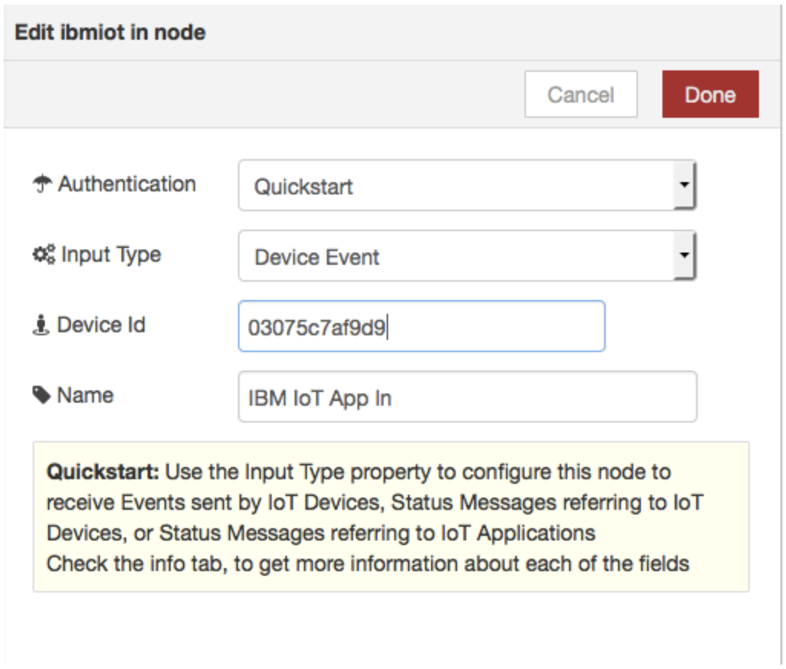
4. After you start your Internet of Things Platform Starter instance, click the following link to connect a temperature simulator to the **IBM IoT app In** node as shown in blue in Flow 1: ibm.biz/iotsensor.



5. Copy the value in the upper-right corner of the virtual thermometer. In the image above, the value is `03075c7af9d9`.
6. In the flow editor, double-click the **IBM IoT App In** node.



7. Paste the ID into **Device Id** field. Click **Done** and then click **Deploy**.



Edit ibmiot in node

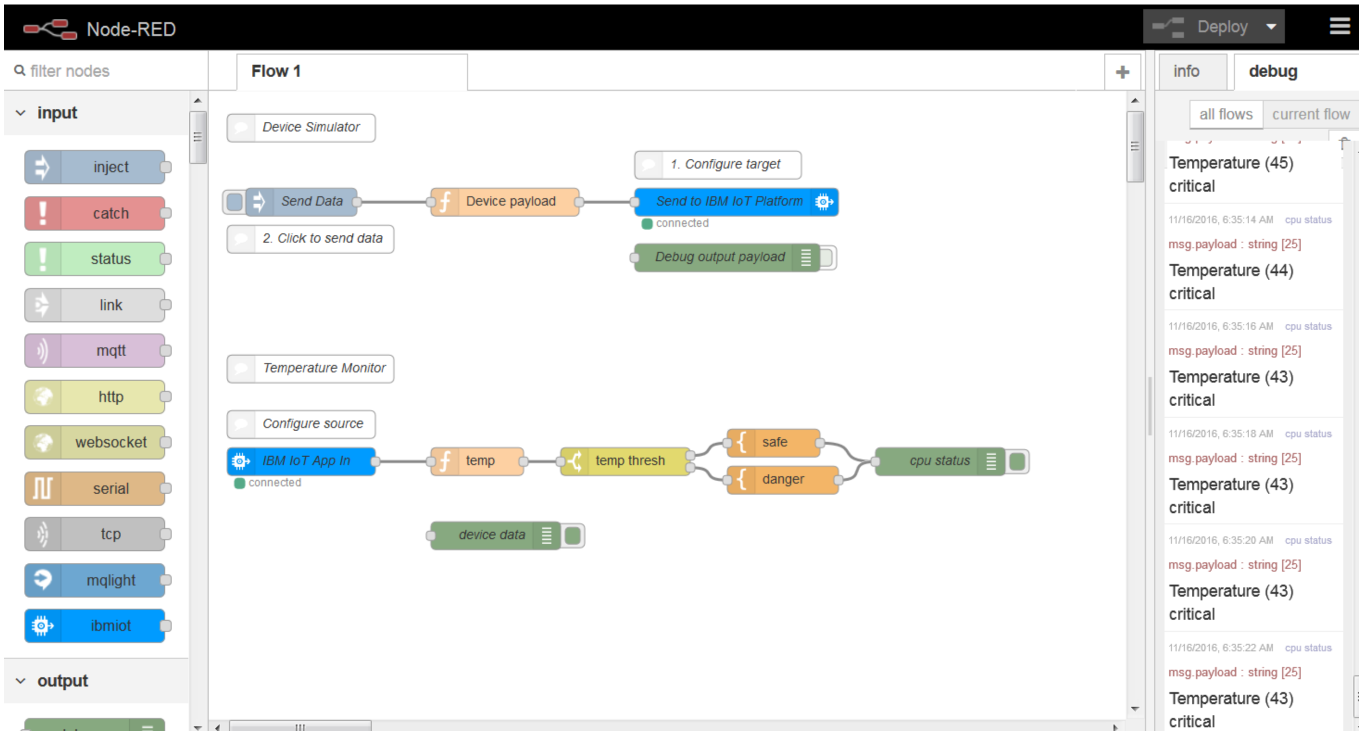
Authentication	Quickstart
Input Type	Device Event
Device Id	03075c7af9d9
Name	IBM IoT App In

Quickstart: Use the Input Type property to configure this node to receive Events sent by IoT Devices, Status Messages referring to IoT Devices, or Status Messages referring to IoT Applications
Check the info tab, to get more information about each of the fields

8. Go to the virtual sensor and increase the temperature to 41° C or more.
9. In the flow editor, note that the flow has three green output **debug** nodes that show flow data in the debug pane.

   Disconnect the top green **Debug output payload** node in the top flow by clicking the connecting line and pressing Delete on your keyboard. Disconnect the **device data** node on the bottom flow as shown in the following image. Then, click **Deploy**.

   You might need to scroll down in the debug pane to see the simplified view of temperatures.



```
Temperature (45)
critical
11/16/2016, 6:35:14 AM   cpu status
msg.payload : string [25]
Temperature (44)
critical
11/16/2016, 6:35:16 AM   cpu status
msg.payload : string [25]
Temperature (43)
critical
11/16/2016, 6:35:18 AM   cpu status
msg.payload : string [25]
Temperature (43)
critical
11/16/2016, 6:35:20 AM   cpu status
msg.payload : string [25]
Temperature (43)
critical
11/16/2016, 6:35:22 AM   cpu status
msg.payload : string [25]
Temperature (43)
critical
```

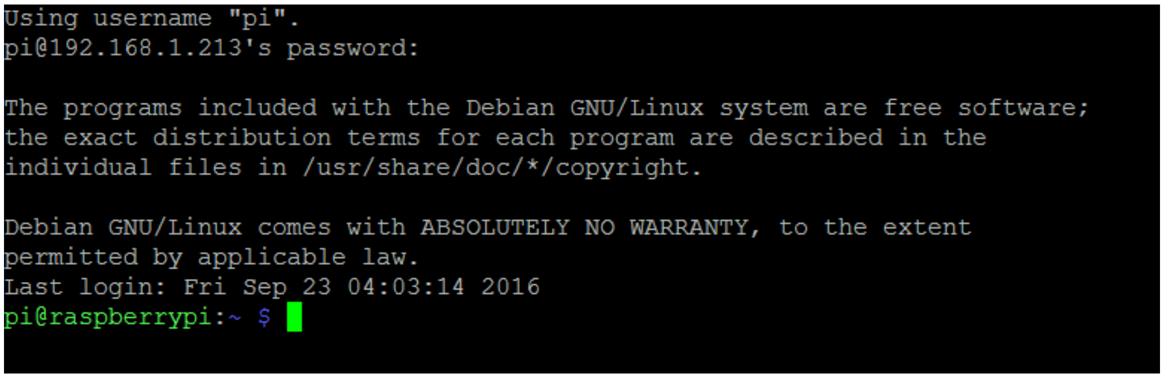
Step 2. Prepare the Raspberry Pi to use as an input source

Be sure you already set up your Raspberry Pi device. For help with setting up the device, see the Help videos and the Raspberry Pi Software Guide.

1. Turn on the Raspberry Pi device with the Raspbian Jessie.
2. Connect all the peripherals monitor, keyboard, mouse, and a mini USB smartphone or digital camera charger for power.
3. Configure the device network to use the same LAN that you are using. **Suggestion**: Use an access point that you might reuse in different locations.
4. Set up networking and launch the Node-RED service. When you launch a terminal, get the IP address that is assigned to your Raspberry Pi by running this command: `ifconfig`
5. Read and save the assigned IP address. In typical LAN locations, it is something like `192.168.1.213`.

   When the IP address is known and if your computer is on the same LAN network, you are ready to launch a terminal on your computer.
6. Run the following Secure Shell (SSH) command and enter the assigned IP address in place of the example IP address: `ssh pi@192.168.1.213`

   If the command is unknown or you have problems with SSH, see Configuring SSH.
7. When you see the prompt for the password, enter `raspberry` and press Enter. (This is the standard password.)

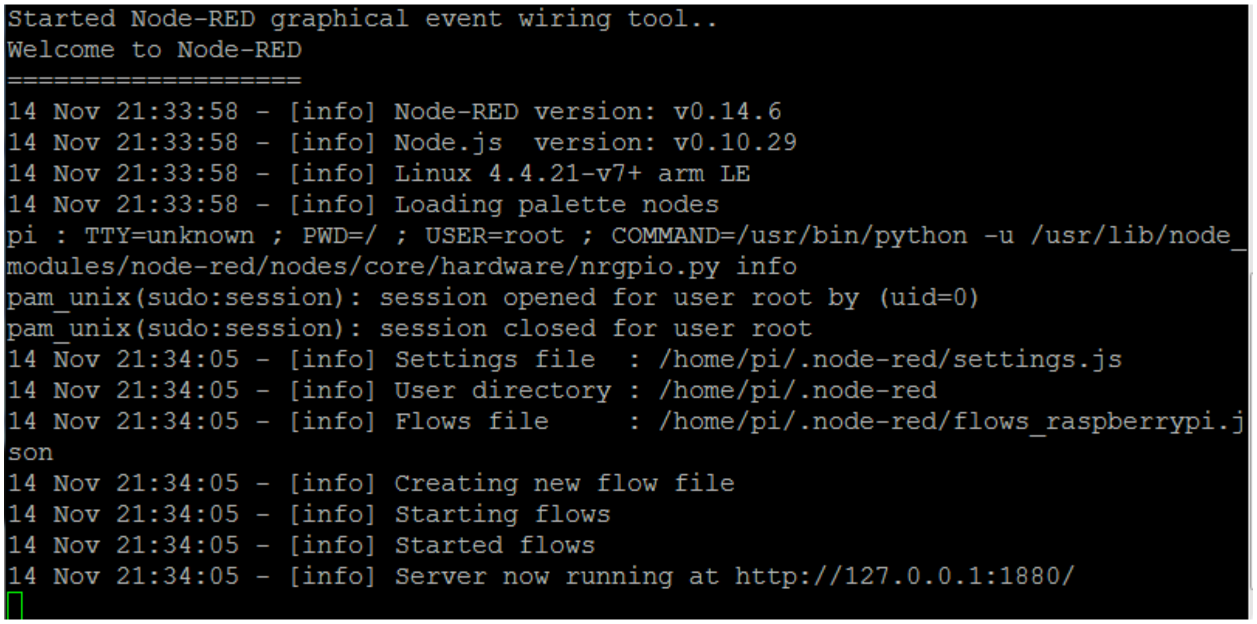


```
Using username "pi".
pi@192.168.1.213's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Sep 23 04:03:14 2016
pi@raspberrypi:~ $
```

8. Start Node-RED. You can start it automatically or manually.
   - Enter this command to automatically start Node-RED:

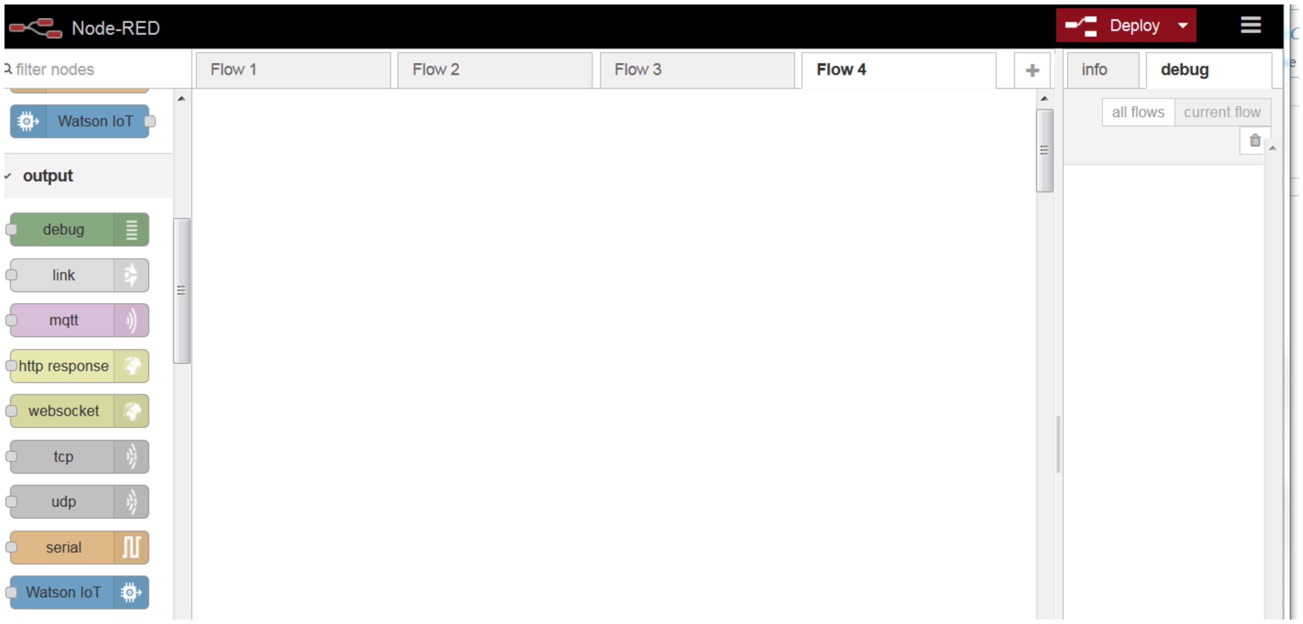
     `sudo systemctl enable nodered.service`
   - Enter this command to manually start Node-RED:

     `node-red-start`



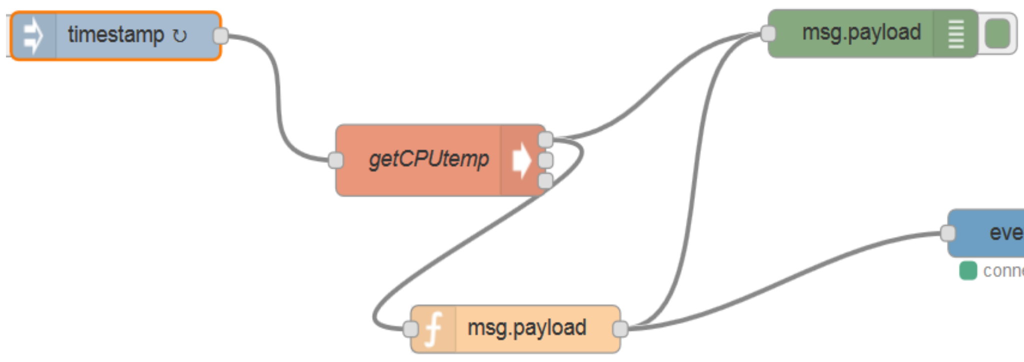
```
Started Node-RED graphical event wiring tool..
Welcome to Node-RED
===================
14 Nov 21:33:58 - [info] Node-RED version: v0.14.6
14 Nov 21:33:58 - [info] Node.js  version: v0.10.29
14 Nov 21:33:58 - [info] Linux 4.4.21-v7+ arm LE
14 Nov 21:33:58 - [info] Loading palette nodes
pi : TTY=unknown ; PWD=/ ; USER=root ; COMMAND=/usr/bin/python -u /usr/lib/node_
modules/node-red/nodes/core/hardware/nrgpio.py info
pam_unix(sudo:session): session opened for user root by (uid=0)
pam_unix(sudo:session): session closed for user root
14 Nov 21:34:05 - [info] Settings file  : /home/pi/.node-red/settings.js
14 Nov 21:34:05 - [info] User directory : /home/pi/.node-red
14 Nov 21:34:05 - [info] Flows file     : /home/pi/.node-red/flows_raspberrypi.j
son
14 Nov 21:34:05 - [info] Creating new flow file
14 Nov 21:34:05 - [info] Starting flows
14 Nov 21:34:05 - [info] Started flows
14 Nov 21:34:05 - [info] Server now running at http://127.0.0.1:1880/
```

9. Enter the `ifconfig` command to see the IP address of the Raspberry Pi device.
10. Enter the IP address in a browser. You can now see the empty node-RED environment running on the Raspberry PI.

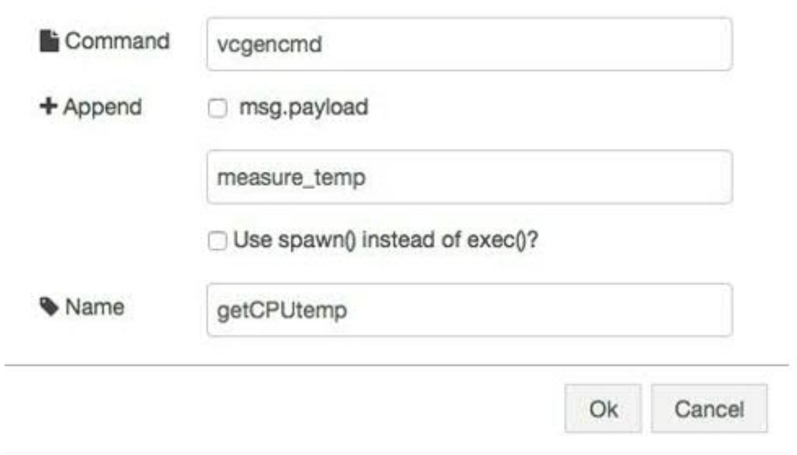


Step 3. Connect the Raspberry PI Node-RED flows to IBM Bluemix

1. Load a new Node-RED flow into the flow editor on the Raspberry PI device:
   a. Copy the raw JSON from this web page to your clipboard: https://raw.githubusercontent.com/ibm-messaging/iot-device-samples/master/node-red/device-sample/quickstart.json.
   b. In the flow editor, create a new flow tab. Then, import the new flow: click the **Menu** icon **> Import > Clipboard**.



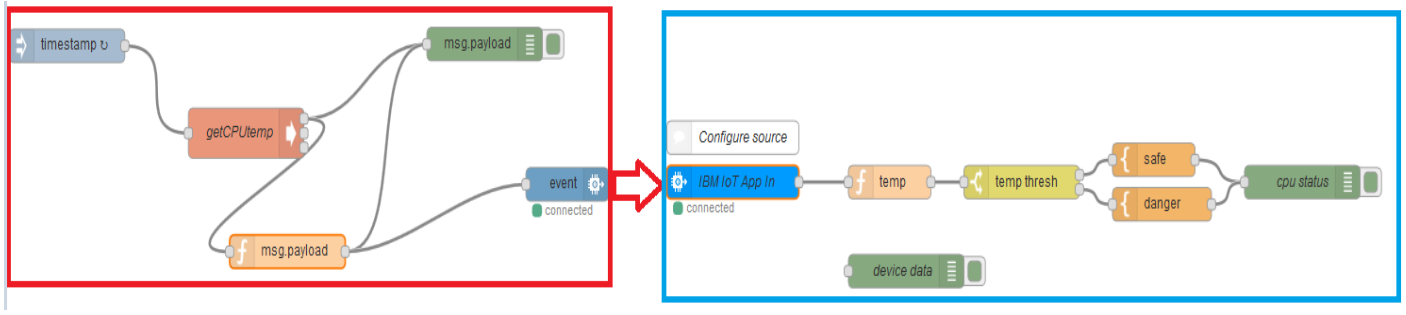
Double-click the red **Exec** node (getCPUtemp), which retrieves the CPU temperature from the Raspberry Pi device. Add the following information:



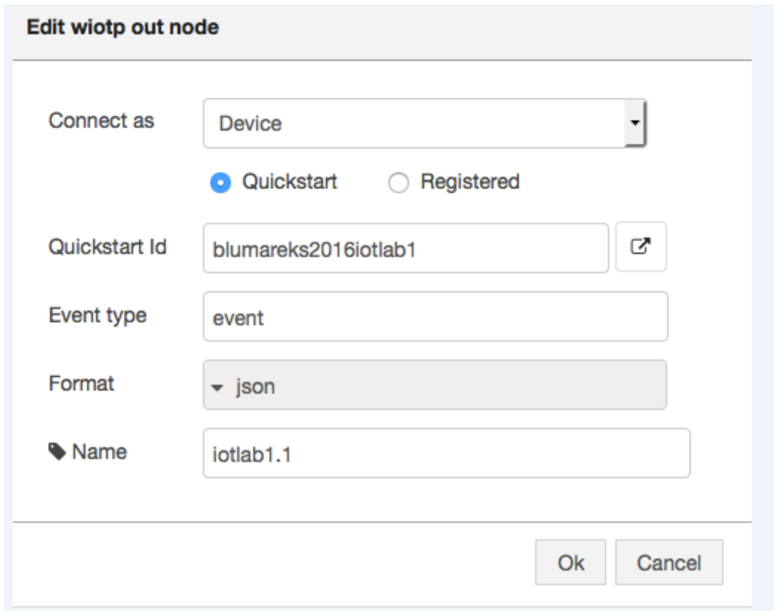
Command	vcgencmd
Append	☐ msg.payload
	measure_temp
	☐ Use spawn() instead of exec()?
Name	getCPUtemp

2. Connect the Node-RED flow on the Raspberry Pi with the Node-RED flow that you already have on Bluemix. The communication between the Node-RED instance running on the

Raspberry PI and the one running on Bluemix is provided by IBM Watson IoT Platform/Quickstart, based on MQTT, which is a lightweight messaging protocol.

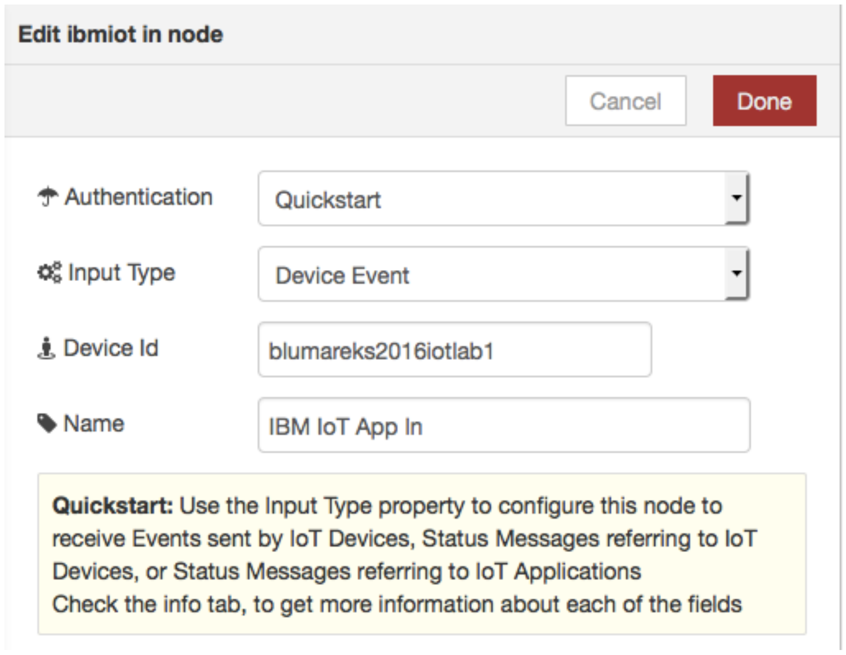


a. Double-click the blue **Watson IoT Output** node (**wiotp out**) in the flow on the Raspberry PI device. Select **Quickstart** and specify a unique value for the Quickstart ID. This value can be any string, but it must be unique in the entire system. Click **OK**.


Edit wiotp out node

Connect as	Device
	[✓ Quickstart] Registered
Quickstart Id	blumareks2016iotlab1
Event type	event
Format	json
Name	iotlab1.1

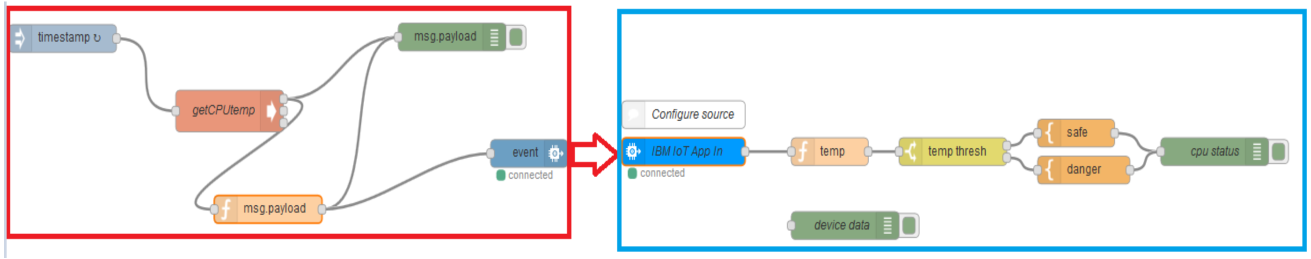
b. Double-click the blue **IBM IoT App In node** and enter the same Device Id that you used for the one on the Raspberry PI. This is how the two Node-RED environments will communicate. Click **Done**.


Edit ibmiot in node

Authentication: Quickstart
Input Type: Device Event
Device Id: blumareks2016iotlab1
Name: IBM IoT App In

Quickstart: Use the Input Type property to configure this node to receive Events sent by IoT Devices, Status Messages referring to IoT Devices, or Status Messages referring to IoT Applications
Check the info tab, to get more information about each of the fields

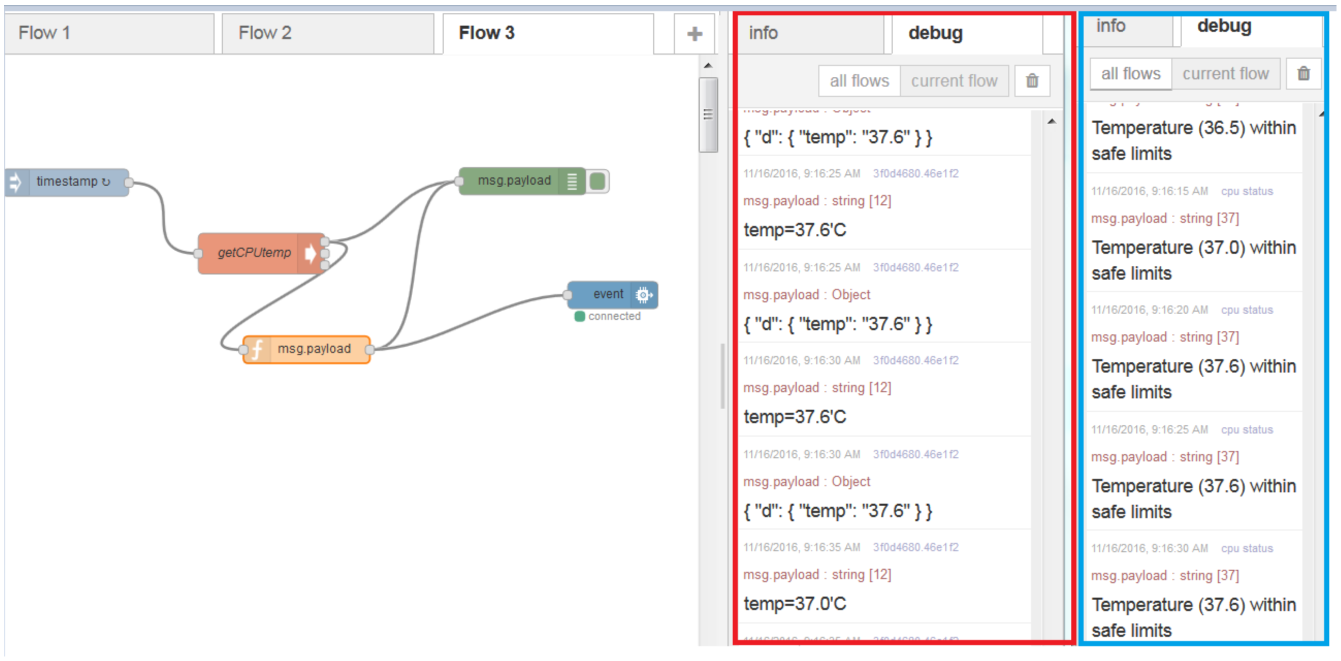
c. Click the **Timestamp** node on the Raspberry Pi to start the flow that will send CPU data to the Node-RED instance on Bluemix every 5 seconds.



d. Review the output in the debug panes.

The two outputs are dissimilar because the Node-RED flow on the Raspberry Pi does not have the trigger warning that you used for the Node-RED flow in Bluemix.

If you disconnect the connections to the green debug node on the Raspberry Pi, you won't see any output in the first debug pane that's highlighted in red, but you will see output in the debug pane highlighted in blue.



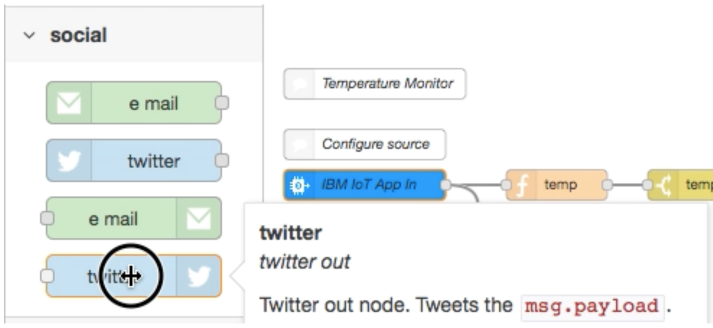
```
{ "d": { "temp": "37.6" } }
11/16/2016, 9:16:25 AM  3f0d4680.46e1f2
msg.payload : string [12]
temp=37.6'C
11/16/2016, 9:16:25 AM  3f0d4680.46e1f2
msg.payload : Object
{ "d": { "temp": "37.6" } }
11/16/2016, 9:16:30 AM  3f0d4680.46e1f2
msg.payload : string [12]
temp=37.6'C
11/16/2016, 9:16:30 AM  3f0d4680.46e1f2
msg.payload : Object
{ "d": { "temp": "37.6" } }
11/16/2016, 9:16:35 AM  3f0d4680.46e1f2
msg.payload : string [12]
temp=37.0'C
```

```
Temperature (36.5) within safe limits
11/16/2016, 9:16:15 AM  cpu status
msg.payload : string [37]
Temperature (37.0) within safe limits
11/16/2016, 9:16:20 AM  cpu status
msg.payload : string [37]
Temperature (37.6) within safe limits
11/16/2016, 9:16:25 AM  cpu status
msg.payload : string [37]
Temperature (37.6) within safe limits
11/16/2016, 9:16:30 AM  cpu status
msg.payload : string [37]
Temperature (37.6) within safe limits
```

Step 4. Add social service notifications to your flow

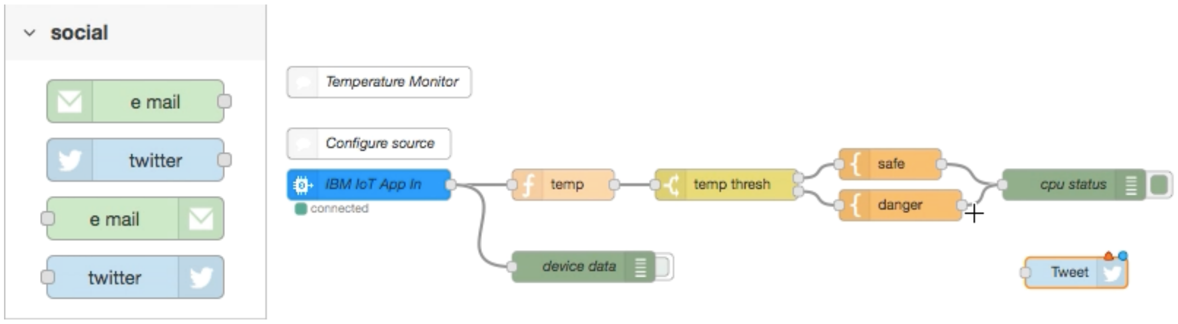
You can easily add other services on Node-RED on Bluemix, such as Twitter, Cloudant database, or a language translator service. In this step we are going to add social service notifications thru Twitter.

The first step is to select on the left panel a Twitter outgoing service (the Twitter icon on the right) as shown below.

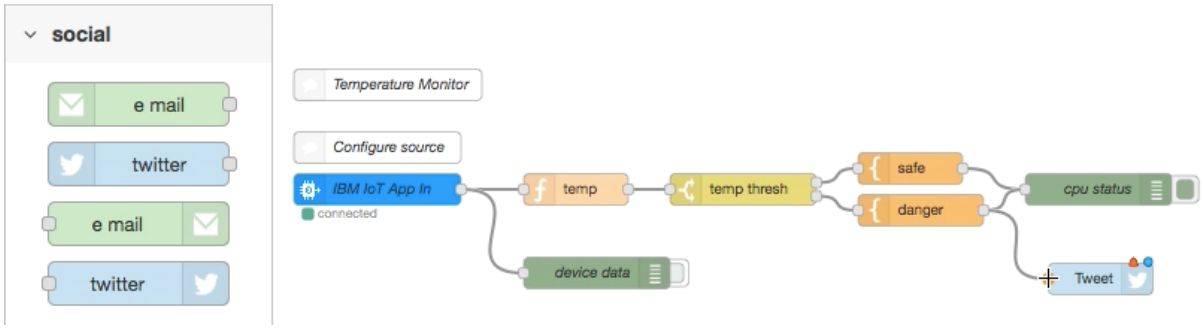


twitter
twitter out
Twitter out node. Tweets the `msg.payload`.

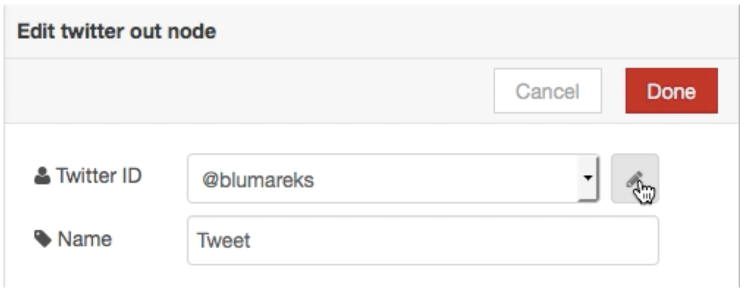
Then drag-and-drop the service in the appropriate place. I decided to inform my followers when the temperature of our imaginary greenhouse (as it was described in the intro) exceeds 40 degrees Celsius.



You need to click the connecting port of the “danger” node on the right and connect it with the Twitter output node.



Double clicking on the Twitter node opens the definition of the service:

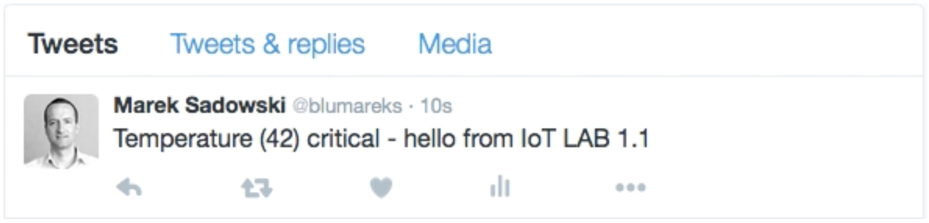


Twitter ID: @blumareks
Name: Tweet

Click the pencil to edit and associate the Twitter node with the Twitter account of your choice.

Hit “Done” on the “Edit twitter out node” and then re-“Deploy” the entire Node.Red flow.

Whenever the temperature exceeds 40 degrees Celsius the tweet would be generated (note that doubled tweets would be filtered by Twitter). See the 42 degrees Celsius tweet generated by our IoT system! Now your followers could let you know that something is wrong with the temperature in your imaginary greenhouse!



Marek Sadowski @blumareks · 10s
Temperature (42) critical - hello from IoT LAB 1.1

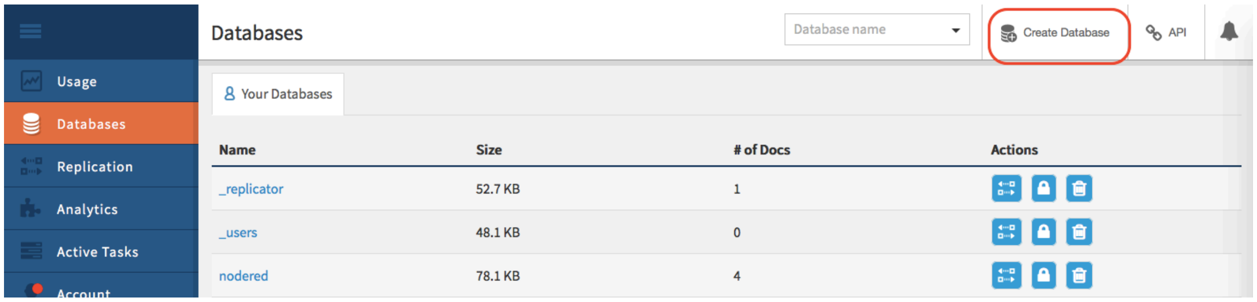
The next step could involve saving the temperatures in the Cloudant DB by adding a Cloudant DB insert node to the flow. Operations on Cloudant DB would be discussed in the next lab.

Experiment with your flow by adding this or other nodes.

Step 5. Add the Swift iOS App that reads the temp

In addition to Twitter you can easily add a Cloudant database node to insert the temperatures into database, and then you might want to read them from your smart device, when you want to validate what is happening with your physical temperature sensor (and follow up with your social network followers from the previous Step 4).

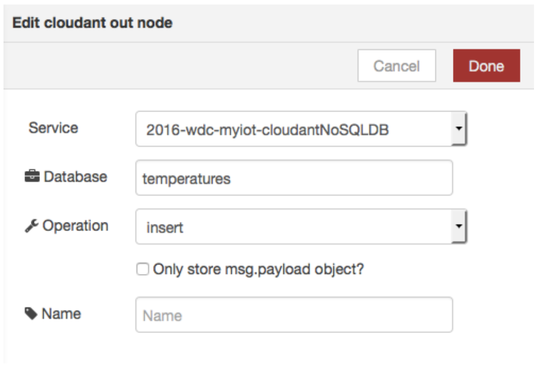
Let's first add the Cloudant database called "temperatures" to the existing Node.Red internal Cloudant DB – from the dashboard just select Databases and hit `Create Database`



Databases

Name	Size	# of Docs	Actions
_replicator	52.7 KB	1	
_users	48.1 KB	0	
nodered	78.1 KB	4	

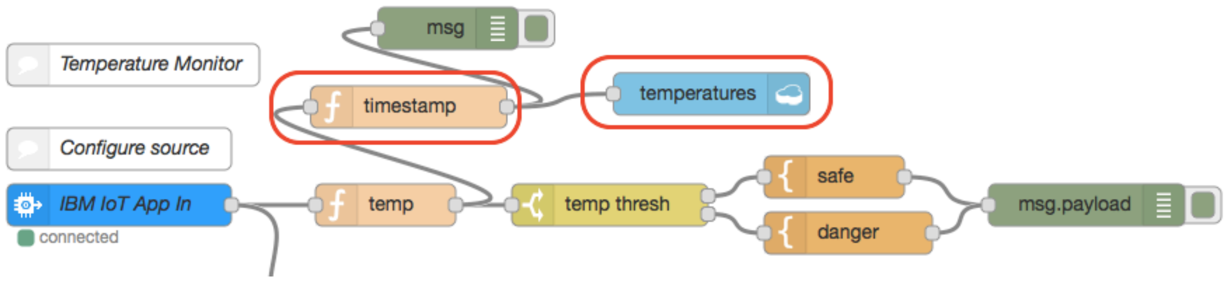
then in the Node.Red flow add Cloudant insert node, configure it to insert data into the internal Cloudant DB with all the msg parameters as it is shown on the provided screenshot:



Edit cloudant out node

Service	2016-wdc-myiot-cloudantNoSQLDB
Database	temperatures
Operation	insert
	☐ Only store msg.payload object?
Name	Name

and connect the Cloudant node and function nodes to the previously created Node.Red flow at the output of the temp node as shown:



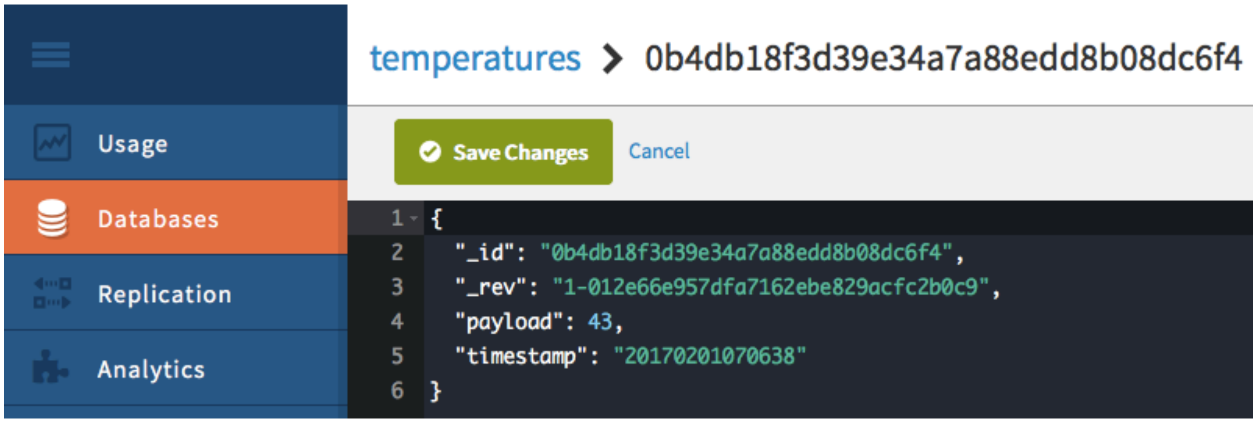
Let's configure the timestamp node to add the time to the temperature in the msg.payload to insert it in the Cloudant DB.

msg.timestamp= new Date().toISOString();

return msg;

Optionally you might want to change it further for the readability purposes. When you are ready to deploy the changes – please re-Deploy the Node.Red flow.

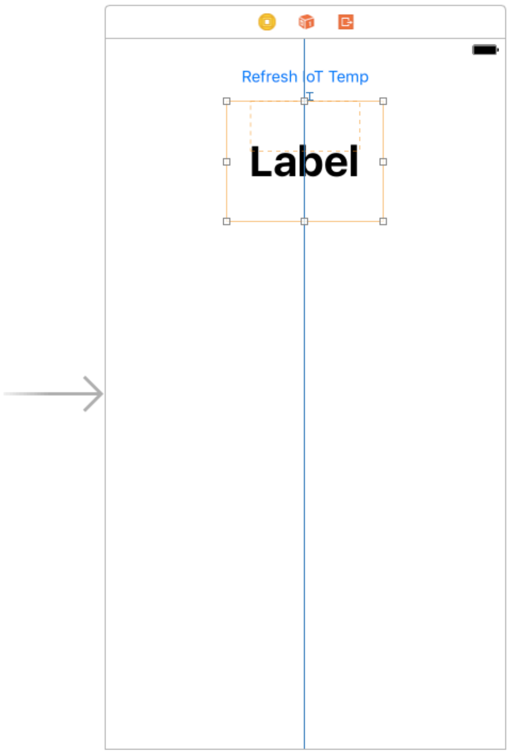
Let’s see how the temperatures are being recorded. We can open the Cloudant DB Dashboard and investigate the temperature records:



```
{
  "_id": "0b4db18f3d39e34a7a88edd8b08dc6f4",
  "_rev": "1-012e66e957dfa7162ebe829acfc2b0c9",
  "payload": 43,
  "timestamp": "20170201070638"
}
```

If you can see the records as shown, we are ready for the next step of building an iOS app with the UI to read the temperature from Cloudant DB.

Let’s create a UI in SWIFT in XCode. Please refer to the solution video on how to deploy a button (Refresh IoT Temp) and the Label on the UI Main.storyboard from the Object Library.



You can now connect the UI with the code. Please refer to the solution video on how to do it.

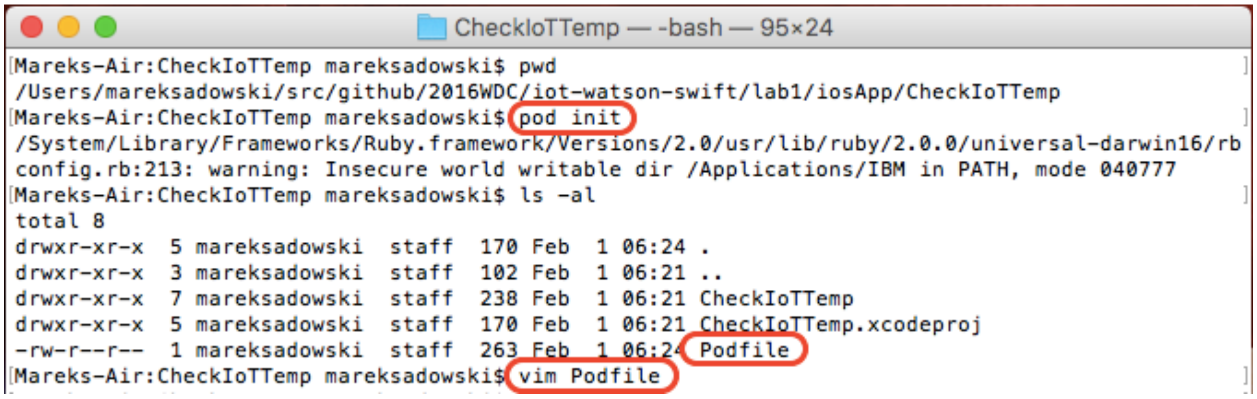
Here comes the final result of doing so:

```
import UIKit
//import SwiftCloudant from CocoaPods

class ViewController: UIViewController {

    @IBOutlet weak var tempLabel: UILabel!
    private var jsonTemp = 0.0

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    @IBAction func refreshButtonPressed(_ sender: Any) {
        NSLog("refresh IoT Temp button pressed")
        //get the temp
        //update the label
        tempLabel.text = "T= ?"
    }

}
```

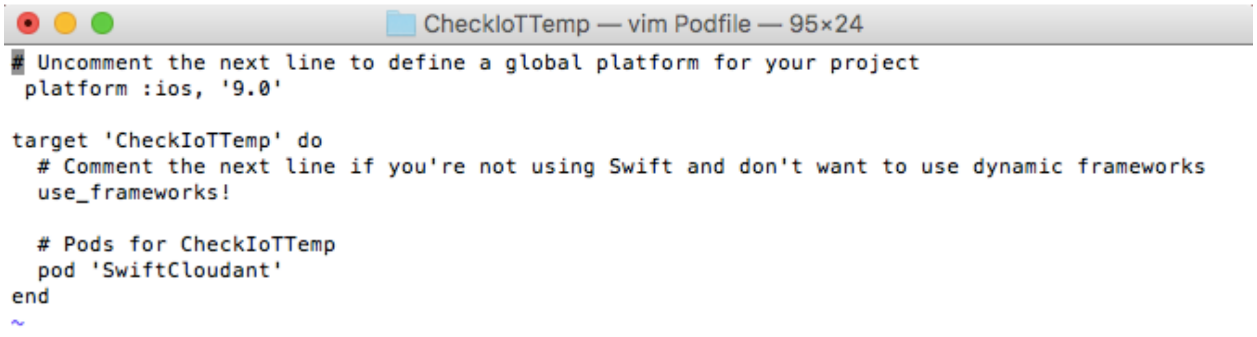
When it is done before the next step we need to close this project (or entire XCode).

Let’s add the SwiftCloudant CococaPods resource. You should get the CocoaPods with the latest XCode, to get the latest update just type in the terminal: `sudo pod update`

You need to install the CocoaPods’ SwiftCloudant pod. In order to do it you need to open terminal, go to the project directory selected when creating an iOS app and initialize `Podfile` by issuing a command `pod init` in the root directory of the app. When it is done you can modify Podfile with the `vim Podfile` command:

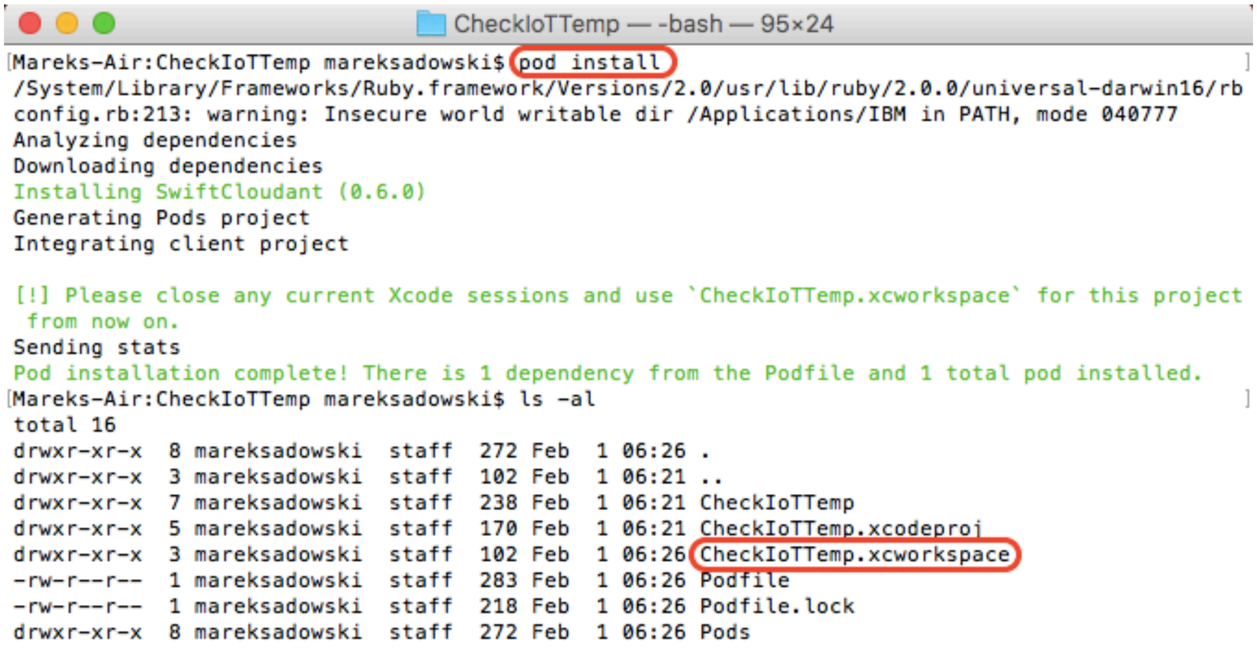


```
Mareks-Air:CheckIoTTemp mareksadowski$ pwd
/Users/mareksadowski/src/github/2016WDC/iot-watson-swift/lab1/iosApp/CheckIoTTemp
Mareks-Air:CheckIoTTemp mareksadowski$ pod init
/System/Library/Frameworks/Ruby.framework/Versions/2.0/usr/lib/ruby/2.0.0/universal-darwin16/rb
config.rb:213: warning: Insecure world writable dir /Applications/IBM in PATH, mode 040777
Mareks-Air:CheckIoTTemp mareksadowski$ ls -al
total 8
drwxr-xr-x  5 mareksadowski  staff  170 Feb  1 06:24 .
drwxr-xr-x  3 mareksadowski  staff  102 Feb  1 06:21 ..
drwxr-xr-x  7 mareksadowski  staff  238 Feb  1 06:21 CheckIoTTemp
drwxr-xr-x  5 mareksadowski  staff  170 Feb  1 06:21 CheckIoTTemp.xcodeproj
-rw-r--r--  1 mareksadowski  staff  263 Feb  1 06:24 Podfile
Mareks-Air:CheckIoTTemp mareksadowski$ vim Podfile
```



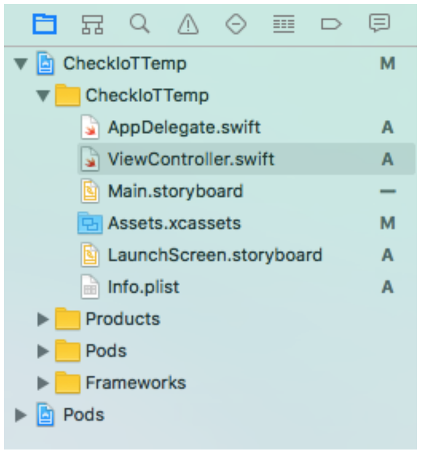
```
# Uncomment the next line to define a global platform for your project
 platform :ios, '9.0'

target 'CheckIoTTemp' do
  # Comment the next line if you're not using Swift and don't want to use dynamic frameworks
  use_frameworks!

  # Pods for CheckIoTTemp
  pod 'SwiftCloudant'
end
~
```

after finishing all edits (uncommenting `platform` and adding `pod ‘SwiftCloudant’`) you can hit ‘esc’ key, and type vi editor command `:wq` that would save the file and close the vim editor. Now you are ready to start the Cocoapods package manager by issuing command `pod install`:



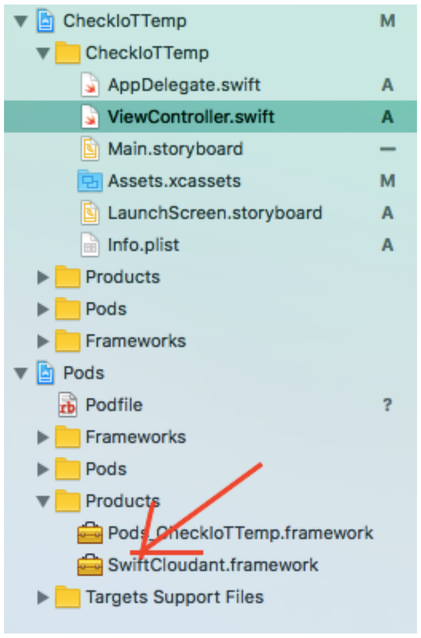
```
Mareks-Air:CheckIoTTemp mareksadowski$ pod install
/System/Library/Frameworks/Ruby.framework/Versions/2.0/usr/lib/ruby/2.0.0/universal-darwin16/rb
config.rb:213: warning: Insecure world writable dir /Applications/IBM in PATH, mode 040777
Analyzing dependencies
Downloading dependencies
Installing SwiftCloudant (0.6.0)
Generating Pods project
Integrating client project

[!] Please close any current Xcode sessions and use `CheckIoTTemp.xcworkspace` for this project
 from now on.
Sending stats
Pod installation complete! There is 1 dependency from the Podfile and 1 total pod installed.
Mareks-Air:CheckIoTTemp mareksadowski$ ls -al
total 16
drwxr-xr-x  8 mareksadowski  staff  272 Feb  1 06:26 .
drwxr-xr-x  3 mareksadowski  staff  102 Feb  1 06:21 ..
drwxr-xr-x  7 mareksadowski  staff  238 Feb  1 06:21 CheckIoTTemp
drwxr-xr-x  5 mareksadowski  staff  170 Feb  1 06:21 CheckIoTTemp.xcodeproj
drwxr-xr-x  3 mareksadowski  staff  102 Feb  1 06:26 CheckIoTTemp.xcworkspace
-rw-r--r--  1 mareksadowski  staff  283 Feb  1 06:26 Podfile
-rw-r--r--  1 mareksadowski  staff  218 Feb  1 06:26 Podfile.lock
drwxr-xr-x  8 mareksadowski  staff  272 Feb  1 06:26 Pods
```

Now you can open the XCode and instead of launching xcodeproj you need to use **xcworkspace** as it was described in the output of the Cocoapods project. In this example it is `CheckIoTTemp.xcworkspace` file as shown above. When you open `CheckIoTTemp.xcworkspace` from XCode you would see your project with the additional Pods folders.



Let's use the SwiftCloudant library in the code to access the DB. After opening the workspace, we need to build the code – by pressing `cmd-B` or by going to the `Product` menu and selecting `Build` item – before referring to the SwiftCloudant library.

When your code is built, you would see the `SwiftCloudant.framework` of the CocoaPods `products` realized.



Only then you can start importing the library with `import SwiftCloudant`:

```
import UIKit
//import SwiftCloudant from CocoaPods
import SwiftCloudant

class ViewController: UIViewController {
```

After doing so you can leverage this library to do CRUD operations in the Bluemix based Cloudant. Here comes the code to connect to it, and read the JSON with the embedded temperature:

```
        //connect to Cloudant
        let cloudantUrl = NSURL(string: "cloudant db connection url")
        let cloudantClient = CouchDBClient(url: cloudantUrl! as URL, username: "cloudant db
connection user", password: "cloudant db connection password")
        let database = "temperatures"

        //find the temp record
        let find = FindDocumentsOperation(selector: [:], databaseName: database, fields:
["payload", "timestamp"], limit: 1, skip: 0, sort:  [Sort(field: "timestamp", sort:
Sort.Direction.desc)], bookmark: nil, useIndex: nil, r: 1)
        { (response, httpInfo, error) in
            if let error = error {
                print("Encountered an error while reading a document. Error:\(error)")
            } else {
                print("Read document: \(response)")
            }
        }
        cloudantClient.add(operation:find)
        //update the label
```

Let's parse JSON to get the temperature. Since the response JSON from the find query looks like the following console debug we would need to get to the nested part of the document marked in red:

```
Read document: Optional(["warning": no matching index found, create an index to optimize
query time, "docs": <__NSSingleObjectArrayI 0x608000007e90>(
{
    payload = 40;
    timestamp = 20170201070636;
}
)
])
```

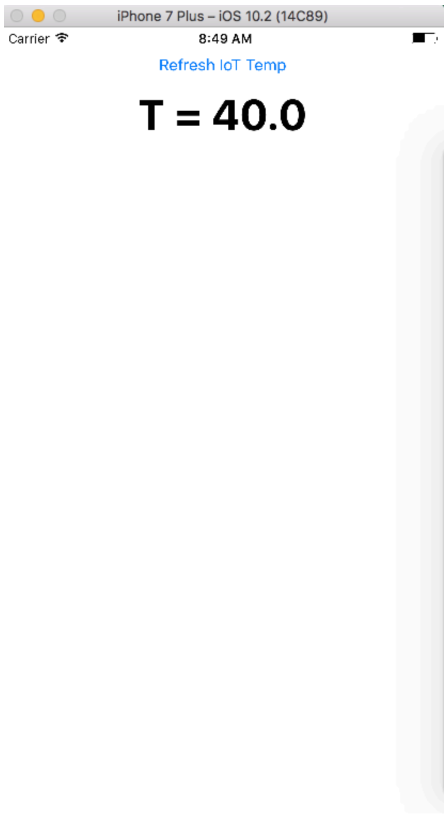
We are interested in getting the `payload` value of the temperature. To parse JSON we use these commands – see the `do – catch clause` used for the `JSONSerialization exception` catching:

```
//get the temp value from JSON
                do {
                    let data = try JSONSerialization.data(withJSONObject: response!,
options: [])

                    let parsedJson = try JSONSerialization.jsonObject(with: data, options:
[]) as! [String:Any]
                    if let nestedArray = parsedJson["docs"] as? NSArray {

                        print("nested \(nestedArray)")
                        //getting nested temp from payload
                        let newDoc = nestedArray[0] as? [String:Any]

                        print("nested \(newDoc)")
                        // access nested dictionary values by key

                        let currentTemperature = newDoc?["payload"] as! Double

                        print("found temp: \(currentTemperature)")
                        self.jsonTemp = currentTemperature

                    }

                } catch  let error as NSError {
                    print(error)
                }
            }
```

Now let's present the results on the UI of the smart device. Therefore, now you need to simply wait asynchronously to get the results sorted out by the SwiftCloudant library:

```
        //update the label
        //we need to wait for the results
        templabel.text = "T= ?"
        DispatchQueue.main.asyncAfter(deadline: .now() + .seconds(1), execute: {
            // Put your code which should be executed with a delay here
            NSLog("Read doc: 1 sec")
            self.templabel.text = "T = \(self.jsonTemp)"
        })
```

And the result looks like this:



The entire code:

```
import UIKit
//import SwiftCloudant from CocoaPods
import SwiftCloudant

class ViewController: UIViewController {

    @IBOutlet weak var tempLabel: UILabel!
    private var jsonTemp = 0.0

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    @IBAction func refreshButtonPressed(_ sender: Any) {
        NSLog("refresh IoT Temp button pressed")

        //get the temp
        //connect to Cloudant
        let cloudantUrl = NSURL(string: "cloudant db connection url")
        let cloudantClient = CouchDBClient(url: cloudantUrl! as URL, username: "cloudant db
connection user", password: "cloudant db connection password")
        let database = "temperatures"
        //find the temp record
        let find = FindDocumentsOperation(selector: [:], databaseName: database, fields:
["payload", "timestamp"], limit: 1, skip: 0, sort:  [Sort(field: "timestamp", sort:
Sort.Direction.desc)], bookmark: nil, useIndex: nil, r: 1)
        { (response, httpInfo, error) in
            if let error = error {
                print("Encountered an error while reading a document. Error:\(error)")
            } else {
                print("Read document: \(response)")

        //get the temp value from JSON
                do {
                    let data = try JSONSerialization.data(withJSONObject: response!,
options: [])

                    let parsedJson = try JSONSerialization.jsonObject(with: data, options:
[]) as! [String:Any]
                    if let nestedArray = parsedJson["docs"] as? NSArray {

                        print("nested \(nestedArray)")
                        //getting nested temp from payload
                        let newDoc = nestedArray[0] as? [String:Any]

                        print("nested \(newDoc)")
                        // access nested dictionary values by key

                        let currentTemperature = newDoc?["payload"] as! Double

                        print("found temp: \(currentTemperature)")
                        self.jsonTemp = currentTemperature

                    }

                } catch  let error as NSError {
                    print(error)
                }
            }
        }
        cloudantClient.add(operation:find)
        //update the label
        //we need to wait for the results
        tempLabel.text = "T= ?"
        DispatchQueue.main.asyncAfter(deadline: .now() + .seconds(1), execute: {
            // Put your code which should be executed with a delay here
            NSLog("Read doc: 1 sec")
            self.tempLabel.text = "T = \(self.jsonTemp)"
        })

    }

}
```

Now we finished LAB 1. The next LAB 2 introduces Watson cognitive services.

Configure SSH

If the SSH commands do not work, follow these steps:

1. Install SSH: `sudo apt-get install ssh.` Wait for the installation to complete.
2. Start the daemon (UNIX name for a service) with this command from the terminal:
   ```
   sudo /etc/init.d/ssh start
   ```
3. Optional: To start the daemon every time you start Raspberry Pi, add the command:
   ```
   sudo update-rc.d ssh defaults
   ```