Build a remote-controlled Raspberry Pi 2 monitor for your IBM Cloud apps

Use Node.js for an Internet of Things project that connects devices and apps through the IBM Watson IoT Platform

Sing Li

April 28, 2015

Demystify the Internet of Things with a hands-on project that uses the IBM Watson IoT Platform to interconnect your devices and apps. Set up a Raspberry Pi 2 to monitor uptime and access time for a running IBM Cloud app, capturing the results in a graphing app that you can deploy to IBM Cloud. And code a desktop application that controls the Raspberry Pi 2 monitor remotely.

Important: The IBM Internet of Things Foundation (IoT Foundation) is now named IBM Watson IoT Platform. This tutorial was written using its previous name and is based on a previous version of the IBM Cloud interface. 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.

Try IBM Cloud for free

Build your next app quickly and easily with IBM Cloud Lite. Your free account never expires, and you get 256 MB of Cloud Foundry runtime memory, plus 2 GB with Kubernetes Clusters. Get all the details and find out how to get started. And if you're new to IBM Cloud, check out the IBM Cloud Essentials course on developerWorks.

Much hype surrounds the Internet of Things (IoT), and with hype comes bandwagon jargon and clouds of FUD (fear, uncertainly, and doubt) that obfuscate simple concepts. Don't be daunted. As a JavaScript developer, you already have all the skills you need to explore the fascinating world of IoT development now. This tutorial helps you get started through a hands-on project.

After a brief introduction to some basic IoT patterns and the IBM Internet of Things Foundation (IoT Foundation), you'll implement two of those patterns. You'll program a functional remote thing—a Raspberry Pi 2 device—to monitor an application hosted on IBM Cloud. And you'll write an app that you can run from your PC command line to turn monitoring on and off remotely on the Raspberry Pi 2. Your devices and apps will connect and communicate via the IoT Foundation. You'll also deploy a graphing app on IBM Cloud that can visualize the collected statistics from your monitor (or monitors; you can deploy as many as you like). You'll do all your coding with Node.js.
To proceed, you need:

- The sample code. **Download** and extract it to your PC now.
- A Raspberry Pi 2.
- A IBM Cloud account linked to your IBM ID.
- The **Cloud Foundry command-line interface (CLI)**.
- Basic familiarity with Linux® command-line operations.
- Git installed and running on your Linux PC.

### Three IoT patterns

What can you do with a network of intelligent communicating devices? In this field of limitless possibilities, three use-case patterns come to mind: monitoring, remote control, and thing-to-thing communication.

#### Monitoring

Figure 1 illustrates the concept of monitoring data collected by IoT things (devices).

**Figure 1. Monitoring over the IoT**

![Monitoring over the IoT](image)

In Figure 1, devices collect data via built-in sensors or other inputs and send them through the Internet for monitoring, collection, storage, and analysis by waiting apps.

**IoT enablers**

It's now technically and financially feasible for an Internet-enabled computer system (system on a chip, or SoC) to be attached to any device (a **thing**). Computers have been in various devices for decades, but only recently have a few groundbreaking IoT enablers emerged:

- SoCs with built-in support for standard communications capabilities (especially WiFi) became available at extremely low cost, enabling inexpensive things to communicate over a network.
- Support for development in high-level modern languages — such as JavaScript, Processing, and Lua — makes it possible for an exponentially larger number of developers to enter the arena.
- MQTT, a proven lightweight yet highly scalable and robust protocol — with almost no barrier to entry for developers of all skill levels — works well both within
Monitoring examples include:

- A motion sensor can count the number of steps a person takes.
- A heart-rate monitor can report a person's heart rate, and a more sophisticated biometric sensor can continually monitor other vital signs. Accumulated data can be used in medical research or additional offline analysis.
- A vehicle or drone can report its geographical position, fuel consumption, acceleration, and other metrics.
- A smart oven can report its current temperature, the internal temperature of the meat being cooked (via a meat thermometer), and the current time-until-meal-done.
- Smart proximity sensors around the perimeter of a house can send their status to trigger a camera, lights, or other alerts.

The example in this tutorial uses a remotely placed Raspberry Pi 2 to monitor the operation of a IBM Cloud-hosted application.

**Remote control**

The second obvious pattern that comes to mind is remote control of devices, as illustrated in Figure 2.

**Figure 2. Remote control via the IoT**

In Figure 2, a remote-control app sends commands through the Internet to one targeted device among many. The target device listens for commands through the Internet and upon receipt executes them, perhaps to control motors, switches, lights, and so on.

Think:

- Turning on your home's heating system from your office PC or from your smartphone
- Driving a LEGO robot car through a contest maze thousands of miles away
• A dispatch app guiding delivery drones between waypoints from warehouses to customers

The example in this tutorial shows how to write an app quickly that turns monitoring on and off in a remotely placed Raspberry Pi 2.

**Thing-to-thing communication**

One not-so-obvious pattern of operations in the IoT is machine-to-machine communications. In this scenario, devices can monitor and control one another, as illustrated in Figure 3.

**Figure 3. Device-to-device communications over the IoT**

In Figure 3, a proxy app assists in device-to-device communications. A device can send messages containing collected data or commands targeted to another device through the Internet to the proxy app. The proxy app then forwards the message to the targeted device back through the Internet for execution or processing.

Imagine:

• A smart power meter that can dim a roomful of light to achieve a user-preset power-consumption goal
• A soil humidity-sensing system that signals the sprinkler to water appropriately
• An automated TV station programmer that broadcasts different-length editions of a TV show depending on real-time-monitored ratings
• A chameleon camouflage panel that takes data from a network of smart ambient sensors and cameras with different points of view to disguise itself dynamically
• A provisioning system for a global app that activates more or fewer computing resources depending on the measured response time gathered from devices installed globally

Implementing device-to-device communication is beyond this tutorial’s scope.

**IoT Foundation in a nutshell**

The IoT Foundation is a cloud that your devices — regardless of how many or how simple — can connect to and communicate with at any time. Figure 4 shows the role played by the IOT Foundation in your connected world of things.
The added value offered by the IOT Foundation includes:

“If your things are connected to the Internet, they can always talk to the IOT Foundation.”

- An always-available network of proxies, brokers, and servers that are connectable via the Internet. If your things are connected to the Internet, they can always talk to the IOT Foundation.
- For things that send monitoring data regularly, practically unlimited storage of data for later summarization and analysis by your own software and services. (By default, an internal service called Historian handles the data storage.)
- Secured authenticated access for every registered device, and for your applications hosted on IBM Cloud or other platforms.
- Scalable, managed, reliable high-speed interconnects among your devices’ entry points, accumulated body of data, and processing resources within the IBM Cloud ecosystem.
- A rich set of tools, templates, and recipes for the rapid creation of monitoring, remote control, or secured proxied thing-to-thing projects.

You can also host and manage your own set of proxies, brokers, and servers. But security, management, and ensuring network and hardware availability can become a daunting full-time endeavor that distracts from your main focus.

In Figure 4, notice that the IoT Foundation consists of a secured MQTT network along with added-value services that can be accessed via HTTP (REST-like) APIs. MQTT (which stands loosely for Message Queue Telemetry Transport) is a lightweight protocol built on top of TCP. MQTT offers simple publish-subscribe semantics for messages exchanged over TCP. Versions of this mature protocol have been evolving for over a decade. In 2013, Version 3.1 was submitted to OASIS, an international standards body. Increasingly, Version 3.1.1 is being deployed in modern global IoT networks. The MQTT client portion can be readily implemented on popular 16- and 32-bit embedded processors with TCP support. A minimal-requirement client can be implemented with very little code overhead, making it ideal for the things end of an IoT network. The protocol itself
also imposes minimal on-the-wire overhead on TCP. The beauty of MQTT is its simplicity. For developers already familiar with the publish-subscribe pattern — such as an `addListener` interface — the MQTT learning curve is zero. The best way to understand the value of MQTT is to work with it, as you'll do shortly.

**A web app to monitor**

The code for the application that you'll monitor (a bare-bones app meant to represent any running IBM Cloud app) is in the sample code's `bluemixwebapp` directory, ready to be deployed to your IBM Cloud environment. The application displays a simple form with two fields, shown in Figure 5.

**Figure 5. Add-item form on the web app that you'll monitor**

![Add-item form on the web app that you'll monitor](image)

Your Raspberry Pi 2 monitor will access this form, fill in the item and quantity fields, and then submit it. The web application will then display a confirmation message (showing item name and quantity), similar to the one in Figure 6.

**Figure 6. Result page on monitored web app**

![Result page on monitored web app](image)

The Raspberry Pi 2 monitor will scrape the result page to validate the confirmation message before reporting the app's status and access time to the IoT Foundation. (I'm using `access time`
to mean the total time to prepare a request, send the request, wait for the response, and scrape. Response time, the most variable value, is primarily accountable for variations in access time.)

From the Cloud Foundry CLI, connect to IBM Cloud, log in, and deploy the app to your IBM Cloud environment:

```
cf api https://api.ng.bluemix.net
cf login -u user_name -o org_name -s space_name
cf push your web app name -m 128M
```

Verify that the web app is up and running by accessing the IBM Cloud web app URL in a browser. The Raspberry Pi monitoring code must know this URL to access the web app, so change the first line in the sample code’s raspi/monitortask.js file to reflect the name:

```
var MonitoredBluemixURL = 'http://your web app name.mybluemix.net/';
```

**Second-generation Raspberry Pi**

Raspberry Pi 2 is the second generation of the world-famous $35 Linux computer. When Eben Upton and friends first set out to make a few affordable computers to help get kids programming, they never dreamed about selling 5 million units in three years and becoming a part of computer history.

The Raspberry Pi 2 is a formidable computing platform capable of running major modern operating systems such as Ubuntu and Windows® 10. Unlike the first-generation Raspberry Pi, the Raspberry Pi 2 features a 900MHz quad-core ARM Cortex-A7 CPU (the original had a single core ARMv6 CPU) and 1GB of RAM (the original only had either 512MB or 256MB). The Raspberry Pi 2 scores 6 to 20 times better than the original in benchmarks. It also has four USB ports for mouse, keyboard, and accessories; a MicroSD card slot for booting OS images; an HDMI port to connect directly to monitors or TVs; and an Ethernet port for network connection. Figure 7 shows the Raspberry Pi 2 on the right, next to the first-generation Raspberry Pi.
Building a Raspberry Pi 2 IBM Cloud app monitor

You're ready to start building a functional monitor for a IBM Cloud-hosted web application on a Raspberry Pi 2. By default, the monitor tries to access the monitored IBM Cloud web application every 60 seconds and measures the access time. You can add any number of these monitors to your IoT via the IOT Foundation (under a grouping called an organization).

Node-RED

Node-RED is a user-friendly, visual drag-and-drop editor for creating Node.js IOT applications. Node-RED can be installed via npm and comes with a large selection of ready-to-run nodes that you can drag and connect to create your own apps.

But as a hard-core JavaScript developer, you likely prefer to work directly with Node.js code without an intervening code generator. The application presented in this tutorial, accordingly, is a direct-to-the-iron JavaScript app that accesses the APIs offered by the IOT Foundation.

Transforming the Raspberry Pi 2 into a remote monitor

You have a choice of operating system images to run on the Raspberry Pi 2, including Debian, Ubuntu, and (soon) Windows 10. Raspbian, the default OS for the Pi, is a customized branch of Debian Wheezy for Pi. Raspbian releases are typically ahead of other OSs in supporting new Pi hardware. The code for this tutorial was developed on the Raspbian dated 2015-02-16.

After you have your Raspbian image up and running, make sure you install the latest available Node.js version on the Pi and on your command center (your Linux PC's console). The sample code is based on Node.js v0.12.0.
Connect from your command center to the Raspberry Pi 2 (via ssh or telnet, for example) and transfer the contents of the sample code's raspi folder (via scp or wget) to your Raspberry Pi 2. The code running on the Raspberry Pi 2 will:

1. Authenticate and connect to the IoT Foundation MQTT network and keep the connection alive.
2. Subscribe to commands using a wildcard iot-2/cmd/+/fmt/+ topic, which enables monitoring to be remotely suspended or resumed.
3. Every 60 seconds, start a controlled browser session, access the IBM Cloud-hosted web application by filling out and submitting a form, and then analyze the results to verify that the web app is working correctly.
4. Publish the access time and status from Step 3 as an accesstime event to the IoT Foundation via the iot-2/evt/itemsvc/fmt/json MQTT topic.
5. If a suspend command is received, stop monitoring the web application for the current cycle.
6. If a resume command is received, resume monitoring in the next cycle.

### Adding Raspberry Pi monitors to your IoT organization

**MQTT client library for Node.js**

Several MQTT JavaScript client libraries are available. The example code uses the MQTT.js module for Node.js — installable via npm install mqtt. This implementation has a stand-alone shell-script-friendly command-line invocation syntax. You can send an event from a device to an unauthenticated MQTT broker/server with a simple command (after you install MQTT globally via the -g option):

```
mqtt publish -t 'devicecmd' -m 'suspend' -h 'test.mosquitto.org' -v
```

Try mqtt publish --help for many more options, including specification of TLS connection, authentication parameters, device ID, and so on.

The fastest way to get the IoT Foundation working for you is through the IBM Cloud dashboard:

1. From your browser, log in to Bluemix and use the SDK for Node.js runtime to create a new Cloud Foundry web app (which will later be replaced by the graphical visualization app). Your app must have a unique name on IBM Cloud.
2. Verify that your newly created app is up and running.
3. Return to the IBM Cloud dashboard. Add an Internet of Things service instance and bind it to your new app. Restage the app if necessary.
4. In your app's overview page, click the Internet of Things service tile.
5. Click Launch dashboard to launch the IoT Foundation dashboard. (You might want to bookmark it for later access.)
6. On the IoT dashboard, you can see your organization name at the top. Note this name, which you'll need later for all the code. Your organization is your grouping of IoT devices and apps.
7. Click the People tab. Click Add Person and add your IBM ID so that you can directly access your organization through this IOT Foundation dashboard (without going through the IBM Cloud dashboard) at any time.
8. Click the **Devices** tab. Add a device of type `rasp2monitor` and device ID `euro001` to add the Raspberry Pi 2 monitor to your organization. Click **Continue**.

9. On the last device-add page, you can see your credentials, including an `auth-token` that your Raspberry Pi 2 will authenticate to the IoT Foundation. Copy down the `auth-token` value. You'll need it later, and it can't be regenerated. Click **Done**.

10. Click the **API Keys** tab and add a new API key. Your key and auth token are displayed as in Figure 8. Make a note of these credentials, which can't be regenerated. The credentials will be used by the remote-control app to suspend or resume monitoring across devices.

**Figure 8. Generating new API credentials with the IoT Foundation dashboard**

![New API Key](image)

**Connecting to the IoT Foundation MQTT network**

MQTT is built on top of TCP. The MQTT connection can use either a TCP or a TLS connection. The monitor code uses TLS:

```
tls://myorg.messaging.internetofthings.ibmcloud.com:8883
```

TLS adds end-to-end encryption to your connections. The data transmitted between the device and the IoT Foundation platform is encrypted the same way that common browser HTTPS traffic is. TLS also verifies that you're indeed connected to the named server. TLS 1.2 and MQTT 3.1.1 are currently supported in the IoT Foundation.

Alternatively, you can use an unencrypted connection via the TCP URL:

```
tcp://myorg.messaging.internetofthings.ibmcloud.com:1883
```

The Raspberry Pi 2 has computing power to spare. But on a minimal device, the additional TLS encryption computation load might be too heavyweight, making unencrypted TCP the only viable option.

Instead of specifying a TCP or TLS URI with a port number, for MQTT over TLS you can also use:

```
mqts://myorg.messaging.internetofthings.ibmcloud.com
```
Or for MQTT over TCP connections:

mqtt://myorg.messaging.internetofthings.ibmcloud.com

**IoT Foundation topic space**

In generic MQTT, a client (be it a device or an app), can publish and subscribe to any *topic* without restrictions. A topic is just a text string in any format. Any *message* (also a text string in any format, often called the *payload*) published to a topic is sent to all subscribers.

IoT Foundation imposes a restricted topic space, different for applications and devices, to secure devices from data leaks (and to prevent spy or rogue devices from stealing data from others). When a device publishes and subscribes to an MQTT topic, it has access to a different topic space than an application.

Figure 9 shows the asymmetric capabilities of a device versus an application when accessing the IoT Foundation.

**Figure 9. Capabilities of apps versus devices on IoT Foundation**

In Figure 9, authenticated applications can send commands to and receive events from any device or app in the organization; however, an authenticated device can only subscribe to its own commands and events.

In the IoT Foundation, you distinguish commands by using topic filters. For example, the topic for an app to subscribe to events from a monitoring device can be:

`iot-2/type/rasp2monitor/id/euro001/evt/itemsvc/fmt/json`

The device type (`rasp2monitor`) and device ID (`euro001`) are both explicitly stated.

An app can use wildcards (`/+`) to subscribe to events originating from multiple devices:

`iot-2/type/+/id/+/evt/itemsvc/fmt/json`

A command topic destined for a device, sent from an application, might look like:

`iot-2/type/rasp2monitor/id/euro001/cmd/operation/fmt/json`
Notice the explicit reference to device type (rasp2monitor) and ID (euro001).

For a device to publish an event, you don't need to specify the device type or ID in the topic, because the IoT Foundation is already aware of them. For example, the euro001 device can publish an event via a topic:

\texttt{iot-2/evt/itemsvc/fmt/json}

Or subscribe to commands targeted for the device with a topic:

\texttt{iot-2/cmd/operations/fmt/json}

In piclient.js, the monitor code subscribes to and checks for a suspend command. But you can readily extend the code to add more commands to cater to your own requirements.

**IoT Foundation message format**

In MQTT, messages (a.k.a. payloads) sent to a topic can be text strings in any format. With the IoT Foundation, you can also use payloads consisting of text strings in any format. But if you conform to the IoT Foundation message format, added-value services such as the Historian service become available to you — because the IoT Foundation can parse your data.

The IoT Foundation message format requires only UTF-8 encoded JSON objects with a top-level node of \texttt{d}. Fields in the \texttt{d} node can be numeric or strings. A \texttt{ts} top-level node containing an ISO8601 timestamp string is optional. In piclient.js, the messages sent to the IoT Foundation conform to the IoT Foundation format. Here is a typical access-time message sent by piclient.js:

```json
{
  "d":{
    "accesstime":2248,
    "status":"ok"
  },
  "ts": "2015-03-27T01:39:47.789Z"
}
```

**Client authentication**

Your Raspberry Pi 2 monitor MQTT client authenticates with the IoT Foundation via an \texttt{auth-token} (see Step 9 of Adding Raspberry Pi Monitors to your IoT organization). This token is generated only once when you first register your device. It is salted and hashed upon generation and cannot be recovered (because the IoT Foundation doesn't store the actual token value). If your credentials are lost or compromised, you can revoke and then reregister your device for a new token — but for the device to connect successfully, you must also get the new token to the physical device (by physically retrieving the device or opening the device for remote access, and then changing the code as instructed in the next paragraph).

On the Raspberry Pi 2, edit the piclient.js file and modify the \texttt{connect()} call to use your own \texttt{auth-token} and organization name (which you saved from Step 6 of Adding Raspberry Pi monitors to your IoT organization):
var client = mqtt.connect(
  'tls://your organization.messaging.internetofthings.ibmcloud.com:8883',
  {
    clientId: 'd:your organization:rasp2monitor:euro001',
    username: 'use-token-auth',
    password: 'your auth-token'
  }
);

Installing a webkit stack

piclient.js uses the monitortask.js module to perform monitoring work. monitortask.js tries to access the web app that is being monitored, via the PhantomJS webkit stack. This tool chain — consisting of PhantomJS, CasperJS, and Spooky — enables Node.js-based scripting and automation of a headless webkit (which is like a browser without the GUI). PhantomJS is the headless webkit (browser engine) with control APIs. CasperJS makes extensive use of these APIs while offering high-level scripting for tests and automation. Spooky bootstraps CasperJS and acts as a remote procedure call (RPC) bridge between Node.js and CasperJS instances.

Before you can use the PhantomJS stack, you must install it on the Raspberry Pi 2. Make sure that you follow this sequence:

1. Install PhantomJS. At the time of writing, ARM Linux binaries are not part of the automated PhantomJS build. Luckily, thanks to Nils Måsén (piksel) of the Raspberry Pi community, a binary for PhantomJS is available. Make sure this binary is in your PATH.
2. Install CasperJS globally with `npm install -g casperjs`. (CasperJS must be installed globally for Spooky to work correctly.) The CasperJS installation will detect that PhantomJS is already installed and will not try to download nonfunctional Intel binaries.
3. Install Spooky by running `npm install` in the directory where piclient.js resides.

Starting the monitor

Now you're ready to start monitoring the running web app:

```
node piclient.js
```

To keep piclient.js running all the time on the Raspberry Pi 2, even surviving system power-off and reboot, you can use a Node.js process manager such as PM2 or forever.

Suspending and resuming monitoring remotely

The code for the command-line tool that suspends (or resumes) monitoring is in the sample code's remotecontrol directory. You can run this tool from your command center (PC) console (not on a Raspberry Pi 2 monitoring device).

Before you can use the tool, you must update the code in remotecontrol/sendcmd.js with your API key credentials (from Step 10 of Adding Raspberry Pi monitors to your IoT organization) and your organization name:
var client = mqtt.connect(  
    'mqtts://your organization.messaging.internetofthings.ibmcloud.com', 
    {
        clientId:'a:your organization:adminapp',
        username:'your API key',
        password:'your API auth token'
    }
);  

The general syntax for running the remote-control tool is:

node sendcmd.js suspend | resume device ID

For example, to suspend monitoring on a running monitor with device I euro001:

node sendcmd.js suspend euro001

And to resume the monitoring on the same device:

node sendcmd.js resume euro001

sendcmd.js connects to the IoT Foundation as an app (that is, it can send commands to all devices in an organization); then it publishes the specified operation command to the specified device.

Graphing access-time statistics from the Historian service

Figure 10 shows a graph of access time from the euro001 monitor in the field. You can see access slowdown in the first data point and a break during which the monitoring was suspended (remotely via sendcmd.js).
This visualization app calls the **IoT Foundation HTTP APIs** to access your IoT organization and devices information and fetch the historical access-time data captured by the Historian service. This Node.js Express app uses **Rickshaw** charting to display graphs.

The app is from the IBM Messaging GitHub repository. You can deploy it to your IBM Cloud environment by following these steps:

2. Change the directory to `iot-visualization` and edit the `manifest.yml` file to use your own app name — the app name from Step 1 of [Adding Raspberry Pi monitors to your IoT organization](#). This change will cause the deployment to overwrite the default app. The app already has your IoT Foundation organization bound to it and API key credentials available from the environment.
3. Run the `cf push` command to deploy your app.

**Conclusion**

In a few hours, you completed a practical IoT project that dovetails with your JavaScript and Node.js skills. Monitors like the one that you built in this tutorial can be sent out into the field
(perhaps globally), to locations where typical users of your IBM Cloud app reside. Then you can periodically monitor and analyze the application's uptime and access time to gain valuable insight into the service levels experienced by your geographically diverse user populations.

With the ability to create, monitor, and control thousands of smart things located anywhere, what new applications can you dream up? That's the quintessential IoT killer-app question. For the first time in history, we can parlay our accumulated development and design wisdom into a new universe of connected physical things — controlling them, interacting with them, and assisting them to perform real-life tasks. In so doing, we can change the world.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample code</td>
<td>cl-raspberry-pi-iot-remote-monitoring-app.zip</td>
<td>13KB</td>
</tr>
</tbody>
</table>
Related topics

- **Raspberry Pi**: Learn all about the Raspberry Pi at the official site.
- **MQTT**: Check out the official MQTT Version 3.1.1 OASIS standard or read the original MQTT V3.1 specification. And get the latest MQTT news at the mqtt.org users community site.
- **MQTT brokers and servers and the client libraries**: Find out about the MQTT brokers, servers, and client libraries available across programming languages and environments.
- **IBM Internet of Things Foundation**: Discover the IoT Foundation from a developer's perspective. Check out the available management and historical data access HTTP APIs in addition to the MQTT-based access.
- **IBM IoT Foundation documentation**: Delve into the IoT Foundation documentation. Here you'll learn the capabilities of an application accessing the IoT Foundation versus those available to a device connected to the IoT Foundation.
- **MQTT.js**: Browse through the rich API offered by the MQTT.js client implementation for Node.js used in this tutorial.
- **PM2 and forever**: See how to keep Node.js programs running even after power-off and reboots.
- **PhantomJS, CasperJS, and Spooky**: Get the popular webkit stack.
- **phantomjs-raspberrypi**: Get a ready-to-run PhantomJS binary from piksel's repository.
- **Raspberry Pi OS images downloads**: Get the latest available OS images available for your Raspberry Pi 2, including the Raspbian distribution.
- **IBM Messaging GitHub repository**: Get the latest update of the IoT Foundation Visualization Dashboard, written in Node.js.
- **Obtain the latest update of the sample source code at the author's GitHub repository.**