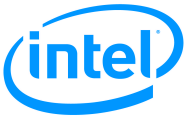
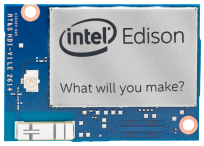
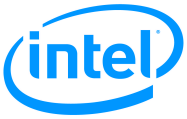




Intel® Cloud-based Analytics Part 1



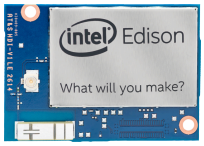
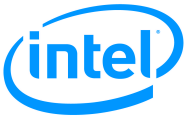
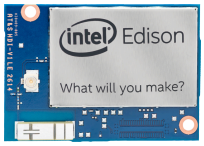


Table of Contents

Revision history		
Version	**Date**	**Comment**
1.0	9/25/2015	Initial release
1.1	10/14/2015	Bug report is included





Introduction

In this tutorial, you will:

1. Be introduced to the cloud-based analytics service,
2. Learn to create an account,
3. Learn to register an Intel Edison and sensors/actuators,
4. Learn to collect sensor data provided by the Edison using IoT agent, and
5. Learn to use custom components such as rotary sensor.

Things Needed

1. An Intel Edison with Arduino-compatible breakout,
2. A micro USB cable,
3. A Grove Starter kit, and
4. A PC or Mac

Intel Cloud-based Analytics Overview

Intel provides a cloud-based analytics system for the Internet-of-Things (IoT) that includes resources for the acquisition and analysis of sensor data that the Intel® IoT Developer Kit provides. Using this service, Intel Galileo/Edison device developers can jump-start data acquisition and analysis without large-scale storage and data processing capacity.

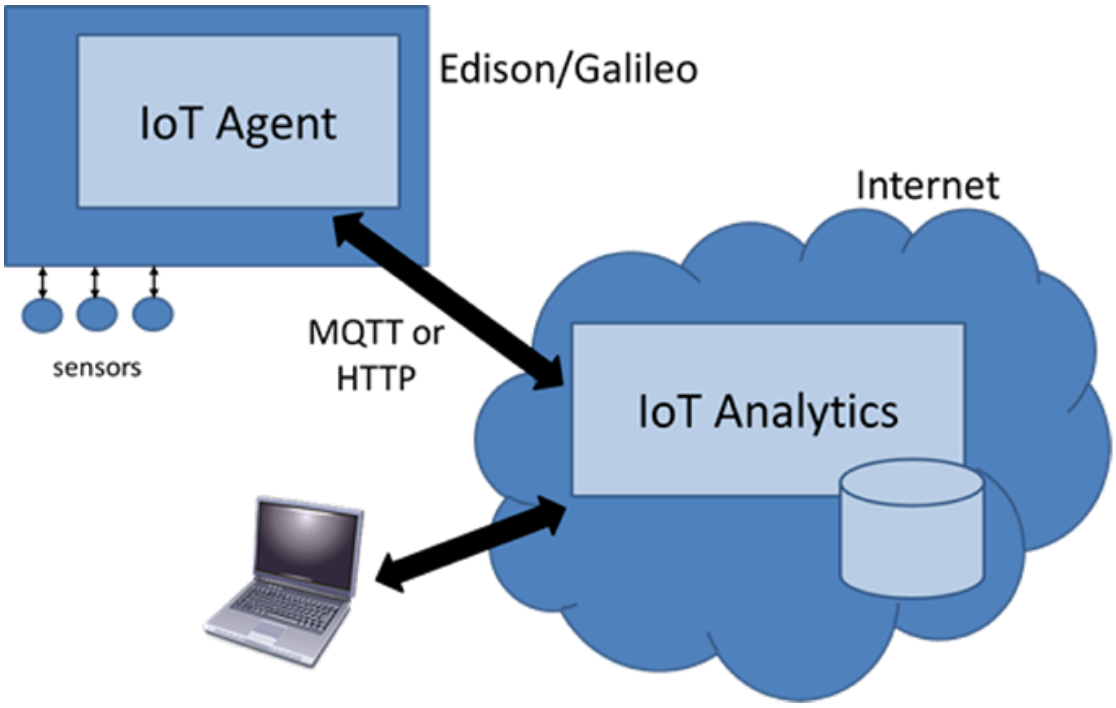


Figure 1 System Workflow of Intel IoT Analytics System (Source: Intel)

As shown in **Figure 1**, the latest embedded Linux image for Edison/Galileo includes a pre- installed IoT Agent that communicates with the Intel IoT Analytics system via either MQTT or HTTP protocols. MQTT is a machine-to-machine/Internet-of-Things protocol that was designed as a lightweight publish/subscribe messaging transport. A user can interact with the IoT Analytics system via IoT Analytics Dashboard as shown in **Figure 2**.

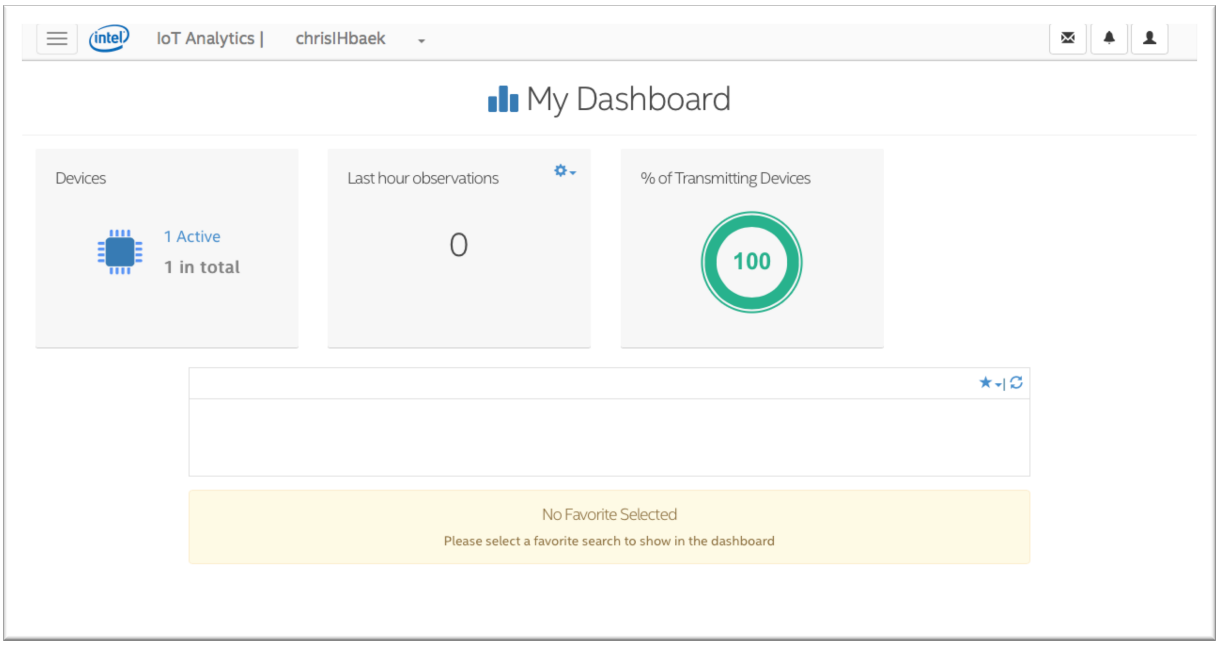
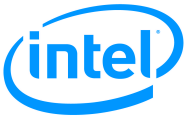


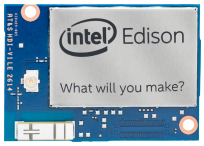
Figure 2 Home Paege of the IoT Analytics System

Before we start working with the Intel Cloud-based Analytics system, please consider the followings:

- Using the iotkit-admin program to change your device ID (from the default of MAC address) does not make the device anonymous. The device IP address will still be available to Intel and the IoT Analytics site. In addition, if a device collects and submits GPS coordinates, Intel will have access to that information, as well. Please carefully review the Terms and Conditions and the Intel Online Privacy Notice.

- Do not connect an Intel® IoT Developer Kit board to any kind of actuator that could cause harm or damage. It is possible that users could unintentionally activate these devices.





Account Creation

Before we proceed, it is important to note that the Intel IoT Analytics Dashboard website makes errors on some web browsers. As of October 14, 2015, the following errors are reported:

Windows

1. Microsoft Edge and Explorer do not interact with the Cloud website properly.
2. Google Chrome is a recommend. All demos are verified on Google Chrome.

Mac OS X

1. Safari has an account verification issue.
2. Google Chrome does not let you create an email notification rule.

In order to use Intel IoT Analytics system, you need to create an administrative account. Follows the steps below to create an account:

1. Go to https://dashboard.us.enableiot.com/
   (Mac users: Do not use Safari at this point, which may have an account verification issue. Once your account is verified, switch to Safari.)
2. Create an account. Although you can log in with your Facebook, Google+, or Github account, some demos in Tutorial 7 (Intel Cloud-based Analytics part 2) will only work with Intel Cloud accounts.
3. A confirmation email will be sent to your email account associated with the Analytics system.
4. Once you verify your account and log in, you will have an access to the IoT Analytics Dashboard as shown in **Figure 2.**
5. The "**Devices**" and the "**% of Transmitting Devices**" box will have a message "**No devices registered yet**" instead.
6. Click on your account name next to IoT Analytics on the top of the page. "chrisIHbaek" in Figure 2 is the account name. Your account name should be shown instead. This will open **My Account** page.
7. There are three tabs, **Details**, **Users**, and **Catalog**.
8. In **Details** tab, you can view account details such as account ID as shown in **Figure 3**. And you can change the account name.

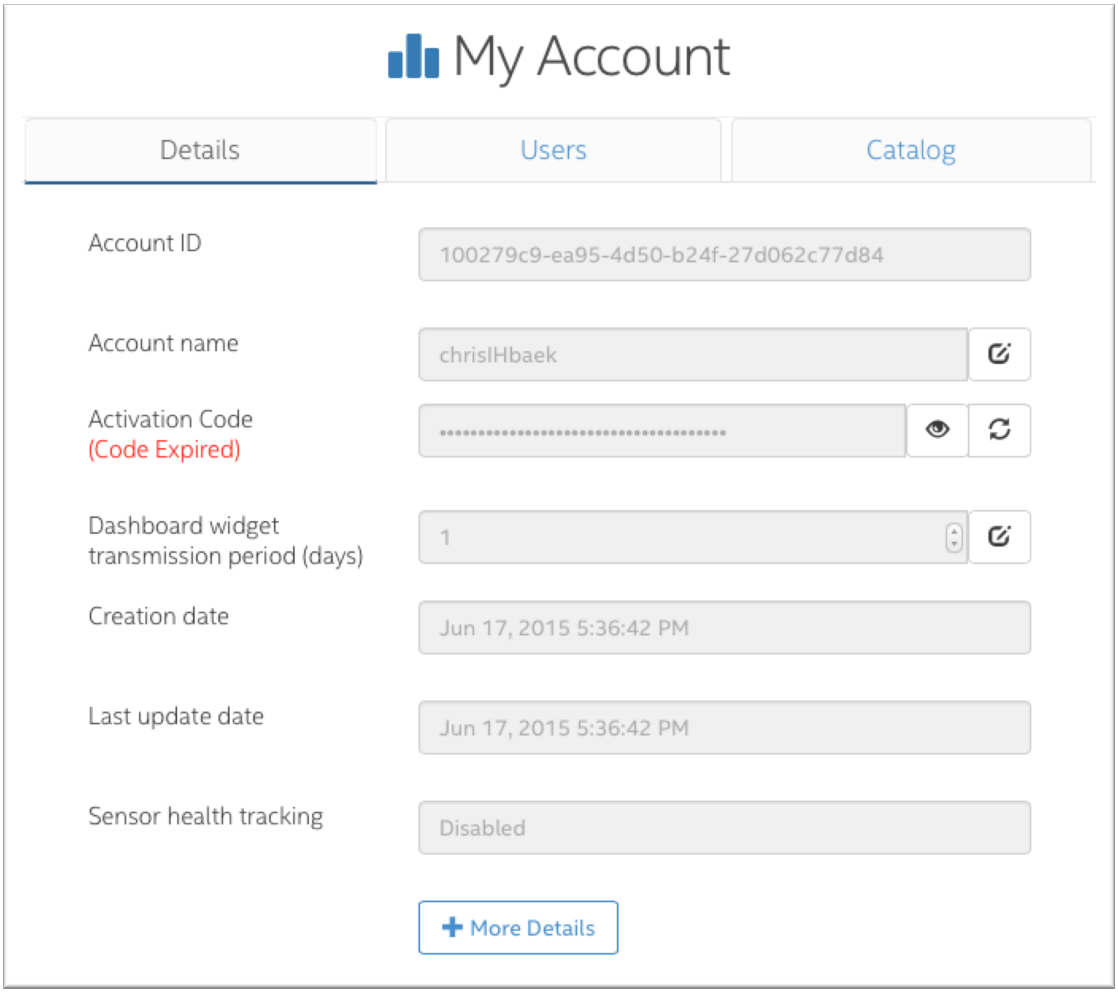


Figure 3 Details Tab

9. In **Users** tab, you may add more accounts to the dashboard and change their roles.

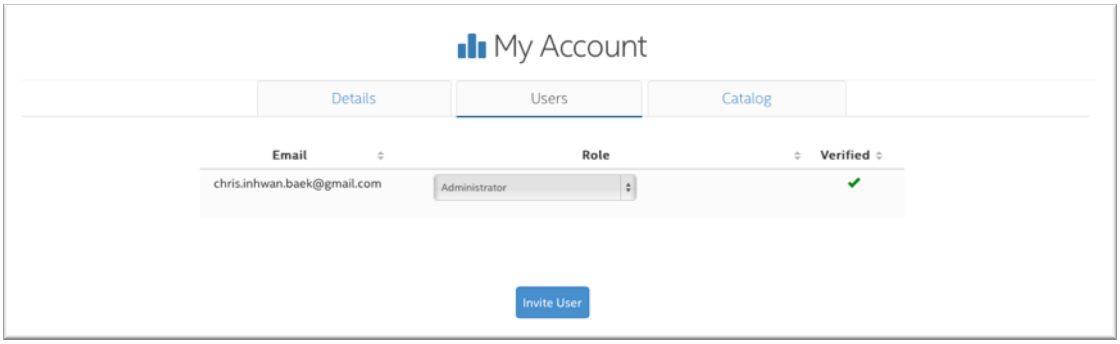


Figure 4 User Tab

10. In **Catalog** tab, you can manage catalog of Component Types. They are templates, which define time series (sensors) and actuators to be used by your devices. You should already have “Humidity”, “Powerswitch”, and “Temperature” by default. Note that “Button” is just a custom Component Type added by the user.

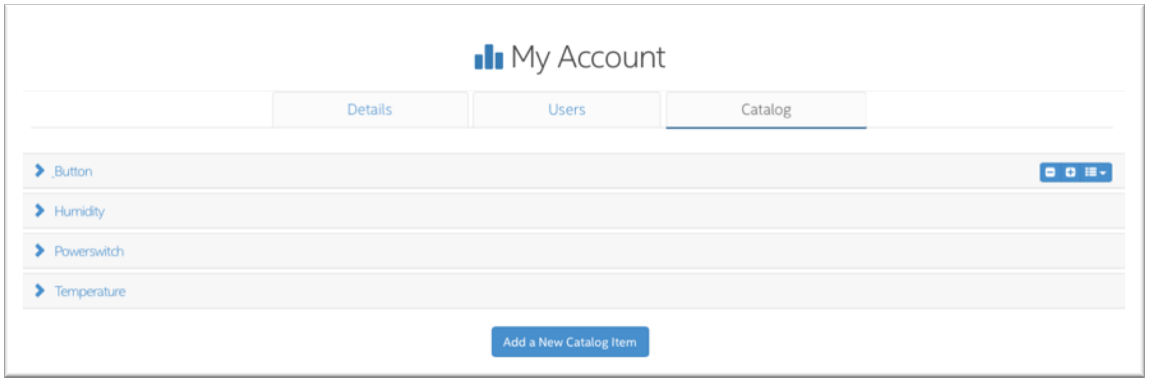


Figure 5 Catalog Tab

The figure below is to help visualize the overall system.

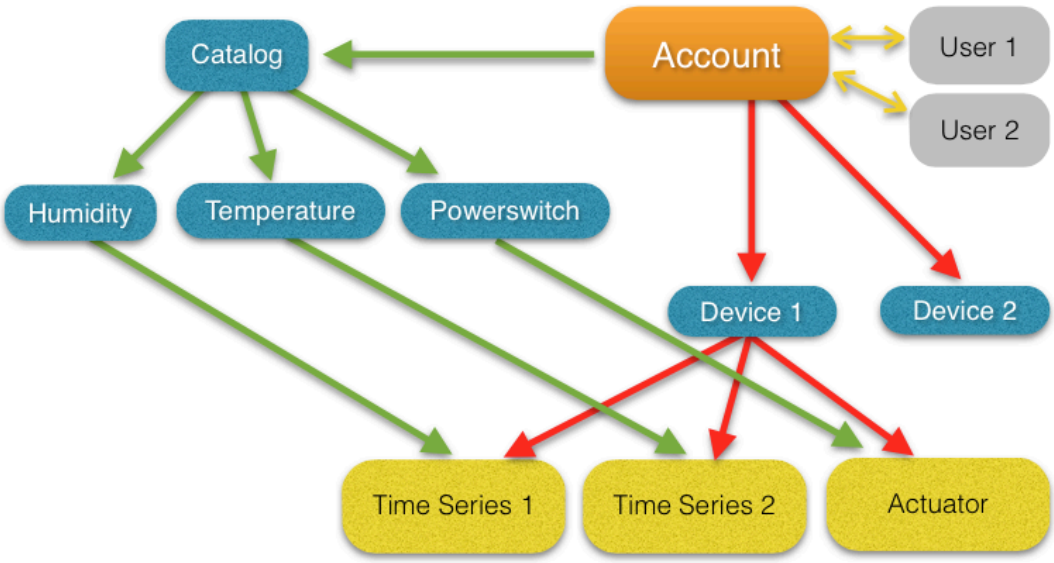
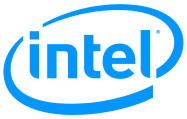
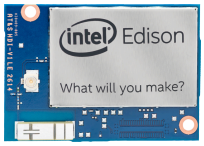


Figure 6 Overall System Diagram





Device and Component Registration

Intel IoT Analytics system supports REST interface that lets the developers write client software. However, the Edison has a pre-installed IoT agent (iotkit-agent) that simplifies the inter-communication between the IoT devices and the IoT Analytics. The IoT agent comes with another program, iotkit-admin, which provides many utility functions, such as testing the network, activating a device, registering time series, and sending test observations.

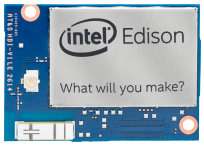
Checking whether pre-installed IoT agent exists

1. SSH into your Edison.
2. **$ iotkit-admin**
3. The following message should be displayed if the agent is installed.

```
root@chris_edison:~# iotkit-admin

  Usage: iotkit-admin [options] [command]

  Commands:

    test                    Tries to reach the server (using the current protocol).
    activate <activation_code> Activates the device.
    register <comp_name> <catalogid> Registers a component in the device.
    reset-components        Clears the component list.
    observation <comp_name> <value> Sends an observation for the device, for the specific component.
    catalog                 Displays the Catalog from the device's account.
    components              Displays components registered for this device.
    initialize              Resets both the token and the component's list.
    update                  Send update device request to dashboard
    pull-actuations         Fetches actuations from last time action was executed or 24h if never and executes them one by one
    protocol <protocol>     Set the protocol to 'mqtt' or 'rest' or 'rest+ws'
    host <host> [<port>]    Sets the cloud hostname for the current protocol.
    device-id               Displays the device id.
    set-device-id <id>      Overrides the device id.
    clear-device-id         Reverts to using the default device id.
    save-code <activation_code> Adds the activation code to the device.
    reset-code              Clears the activation code of the device.
    proxy <host> <port>     Sets proxy For REST protocol.
    reset-proxy             Clears proxy For REST protocol.
    set-logger-level <level> Set the logger level to 'debug', 'info', 'warn', 'error'
    set-data-directory <path> Sets path of directory that contains sensor data.
    reset-data-directory    Resets to default the path of directory that contains sensor data.
    move-data-directory <path> Change directory where data will be stored
    gateway-id              Displays the geteway id.
    set-gateway-id <id>     Overrides the geteway id.
    set-device-name <name> Change device name
    reset-device-name       Resets to default device name.
    set-udp-port <udp_port> Overrides the port UDP listener binds to
    *                       Error message for non valid command

  Options:

    -h, --help           output usage information
    -V, --version        output the version number
    -C, --config [path]  Set the config file path
```

Figure 7 Message Displayed

4. If the message is displayed, stop the IoT agent by entering "**systemctl stop iotkit-agent**". If the agent is not installed, install the agent manually (refer to https://software.intel.com/en-us/node/530661).



Checking network connectivity

1. **$ iotkit-admin test**
2. If the output looks like that in Figure 8, the network connection is good and you can skip to the next subsection "**Registering the Edison**".

```
root@chris_edison:~# iotkit-admin test
2015-07-07T23:06:40.231Z - info: Trying to connect to host ...
2015-07-07T23:06:40.922Z - info: Connected to dashboard.us.enableiot.com
2015-07-07T23:06:40.926Z - info: Environment: prod
2015-07-07T23:06:40.928Z - info: Build: 0.14.2
```

Figure 8 Network Test

3. A firewall may cause a connection issue when using REST protocol.
4. You can use a proxy server by adding the proxy server to the iotkit-admin configuration with a command "**iotkit-admin proxy <host> <port>".**
5. You can clear the proxy by entering "**iotkit-admin reset-proxy**".

Registering the Edison

1. Check your Edison's device ID by entering "**iotkit-admin device-id**"
2. Optional: Although not recommended, you can change your Edison's device ID by entering "**iotkit-admin set-device-id <desired_device_id>".**

```
root@chris_edison:~# iotkit-admin device-id
2015-07-08T00:15:23.889Z - info: Device ID: ee-ab-8c-bd-84-81
```

Figure 9 Device id

Warnings:

1) The device ID must be unique.
2) Any device ID that is taken will prevent device activation.
3) Device activation returns a security token that only works with the device ID used when the device was activated. Therefore, the device ID must not be changed after device activation.

3. On IoT Analytics Dashboard, go to **Details** tab of **My Account** page.
4. Refresh the "**Activation Code**" and make it visible. In **Figure 10**, the activation code is *569qbxnw*.

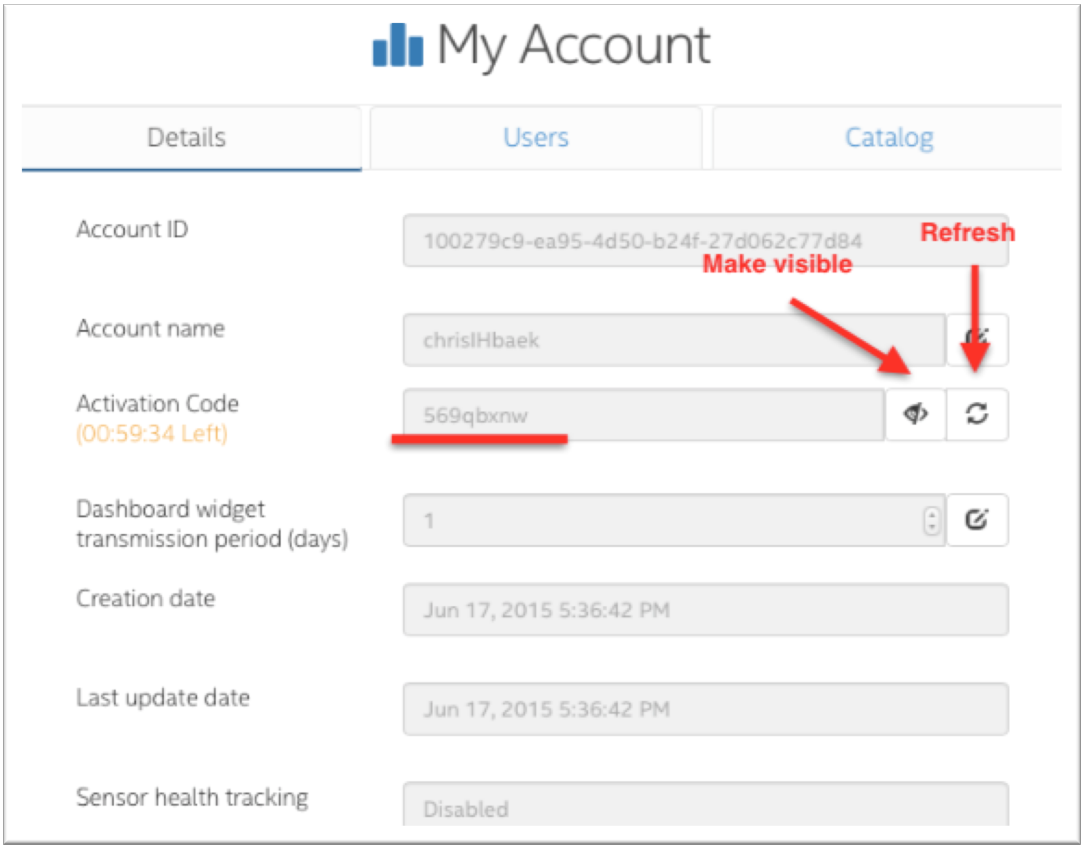


Figure 10 Registration

5. Activate your Edison by entering "**iotkit-admin activate <activation_code>".** Replace <activation_code> with the activation code found in the previous step.
6. On the dashboard, your Edison should be listed on the **My Devices** page.

Figure 11 My Devices

Registering a component

1. **$ iotkit-admin catalog**

```
root@chris_edison:~# iotkit-admin catalog
2015-07-08T00:48:49.212Z - info: Starting Catalog Retrieving
2015-07-08T00:48:50.479Z - info: Comp:  button.v1.0   button   sensor
2015-07-08T00:48:50.483Z - info: Comp:  humidity.v1.0   humidity   sensor
2015-07-08T00:48:50.484Z - info: Comp:  powerswitch.v1.0   powerswitch   actuator
2015-07-08T00:48:50.485Z - info: Comp:  temperature.v1.0   temperature   sensor
2015-07-08T00:48:50.489Z - info: # of Component @ Catalog :  4
```

id : t	kind	measure
button.v1.0	sensor	button
humidity.v1.0	sensor	humidity
powerswitch.v1.0	actuator	powerswitch
temperature.v1.0	sensor	temperature

Figure 12 Catalog

2. The output is the Component Types registered in the Catalog. Note that "button.v1.0" is a custom component registered by the user. You may only have "**humidity.v1.0**", "**powerswitch.v1.0**", and "**temperature.v1.0**" by default.
3. Now let's register a temperature sensor (time series).
4. **$ iotkit-admin register temperature temperature.v1.0**

```
root@chris_edison:~# iotkit-admin register temperature temperature.v1.0
2015-07-08T00:56:12.513Z - info: Starting registration ...
2015-07-08T00:56:12.550Z - info: Device has already been activated. Updating ...
2015-07-08T00:56:12.558Z - info: Updating metadata...
2015-07-08T00:56:12.604Z - info: Metadata updated.
Attributes sent
2015-07-08T00:56:14.993Z - info: Component registered name=temperature, type=temperature.v1.0, cid=bfbb5041-ae3
0-4ce2-8601-f12b7b6d963f, d_id=ee-ab-8c-bd-84-81
```

Figure 13 Sensor Registration

5. This command register a time series that references the Component Type, "**temperature.v1.0**", and create an alias "**temperature**" that is used to send observation data.
6. Let's send some observation data to the IoT Analytics system.
7. **$ iotkit-admin observation temperature 30**
8. **$ iotkit-admin observation temperature 40**
9. **$ iotkit-admin observation temperature 50**

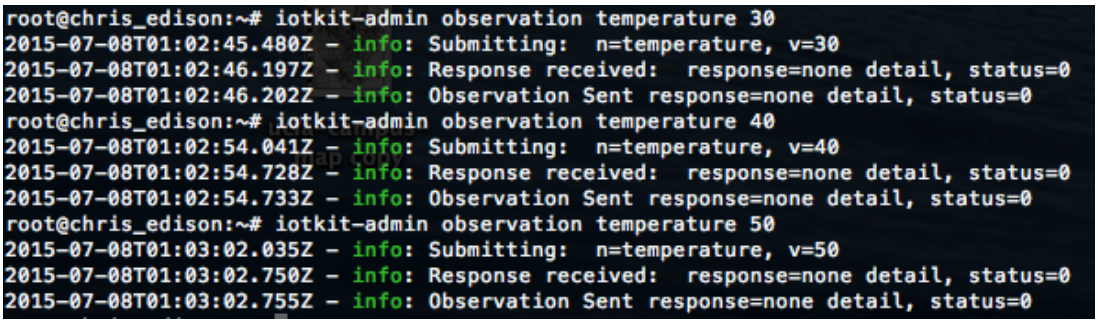


Figure 14 Data Observation

10. Click **Charts** on the sidebar menu of the dashboard.
11. Select your device and select **temperature**.

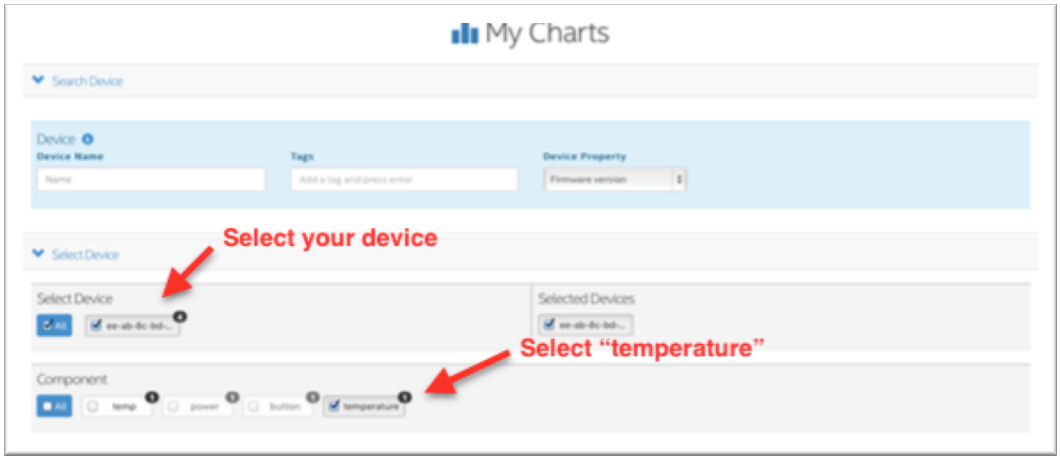


Figure 15 Charts

12. Scroll down to view the received data.

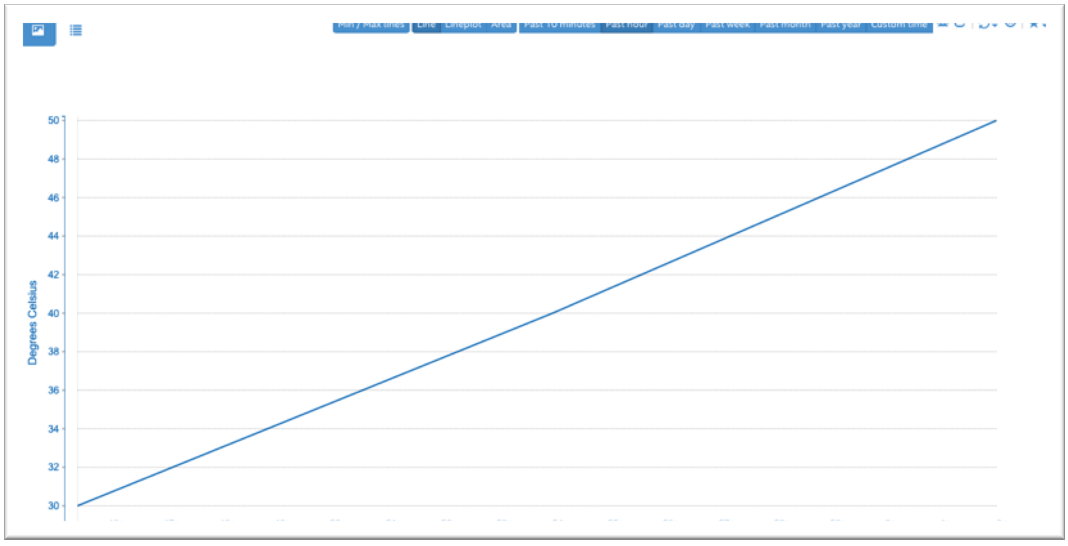


Figure 16 Data Plot

13. As you hover over the line, you can see the exact observation data sent from your Edison.

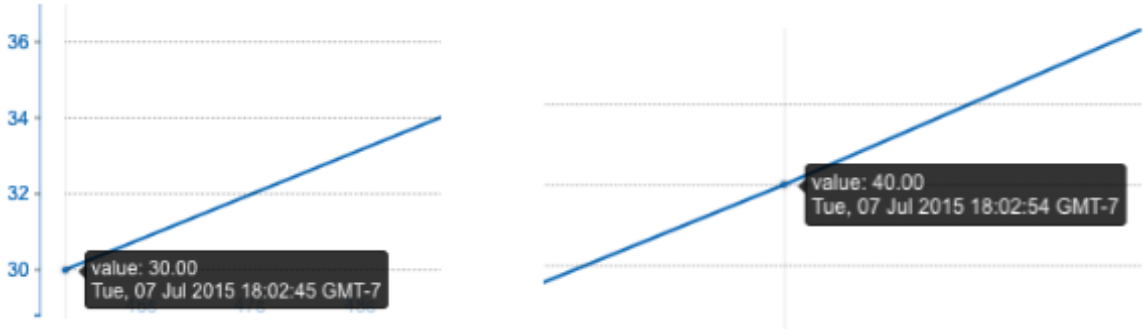
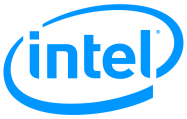
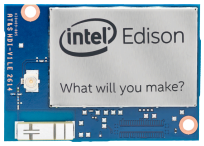


Figure 17 Close-ups





IoT Agent

"**iotkit-admin observation**" command is intended for testing, not for continuous data acquisition. There are two additional methods to send observation data.

1) You can write and use custom client software with REST interface. This client can interact with the cloud directly.
2) You can send User Datagram Protocol (UDP) packets to IoT agent, which converts the packets into REST calls.

In this section, we will focus on method 2. Method 1 will be revisited later.

1. Start the IoT Agent by entering "**systemctl start iotkit-agent**".
2. We will use Intel's official utility called "**send_udp.js**" that will do method 2.
3. **$ mkdir cloud_analytics**
4. **$ git clone https://github.com/enableiot/iotkit-agent.git**
5. **$ cp iotkit-agent/send_udp.js send_udp.js**
6. Let's send an observation with this utility.
7. **$ ./send_udp.js temperature 35.0**
8. Go to **My Charts** page on the dashboard and check whether the cloud received the observation.
9. If not, check whether data submission was done properly by entering "**systemctl status iotkit-agent**".

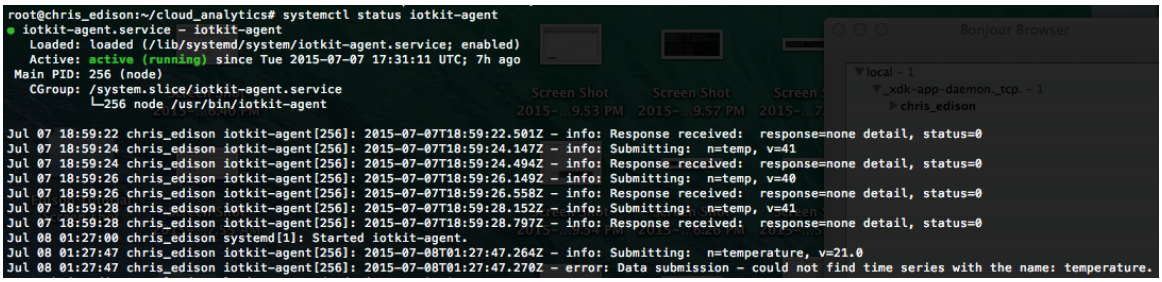


Figure 18 Check Data Submission

10. If the command displays a message "error: Data submission = could not fine time series with the name: temperature.", the IoT agent does not know of the component alias "temperature".
11. We can fix this by restarting the agent.
12. **$ systemctl restart iotkit-agent**
13. Let's send an observation again.
14. **$ ./send_udp.js temperature 35.0**
15. Enter "**systemctl status iotkit-agent**" to check the data submission again.

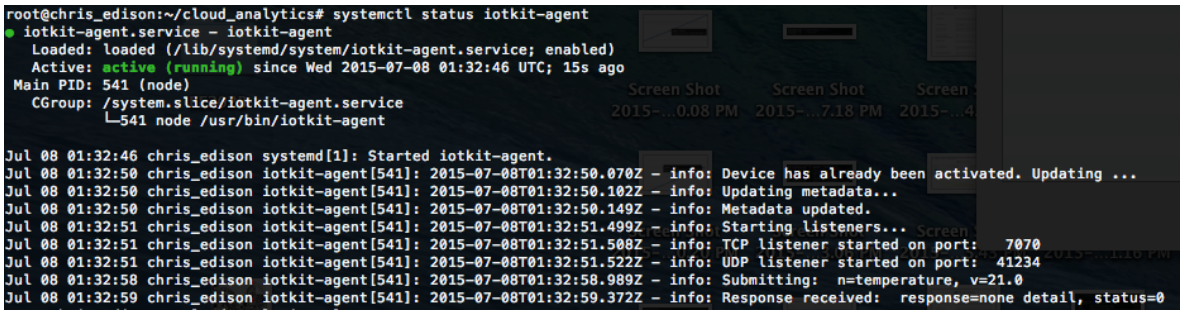


Figure 19 Check Data Submission

16. You should have an output that is similar to the following. The last two lines indicate that the data submission was successful.
17. Now, check on the **My Charts**. The observation must have been received.

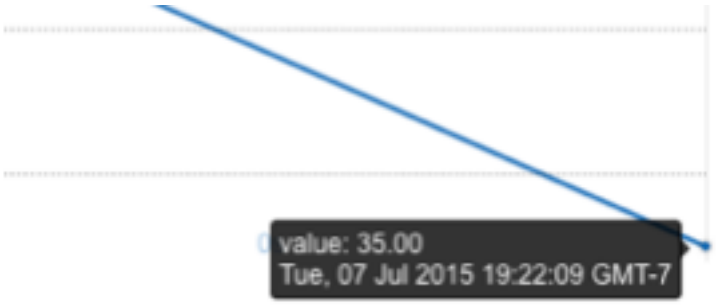


Figure 20 Data Observation

How does it work?

Look at the output of "**systemctl status iotkit-agent**" at step 15. There is a line that states "info: UDP listener started on port: 41234". This indicates that a UDP-formatted message can be sent to the IoT agent via the Edison's localhost on port 41234. Localhost is the hostname meaning "this computer" and can be accessed via the loopback network. For more information, refer to http://whatismyipaddress.com/localhost.

Let's look at the "**send_udp.js**" code. For better readability, rather than entering "cat send_udp.js", read from the following link: https://github.com/intel-iot-devkit/iotkit-agent/blob/master/send_udp.js. In line 30, the code assigns 127.0.0.1, the IP address of the localhost, to a variable called "endpoint", which is then used as a parameter for **client.send** function in line 49. In line 31, 41324 is assigned to a variable called "port", which is also used as a parameter for **client.send** function. "**client**" is a UDP socket created in line 48. For more information on UDP implementation in Node.js, refer to https://nodejs.org/api/dgram.html.

The message sent to the UDP socket is in JavaScript Object Notation (JSON) format, which is **{ "n": "<sensor name>", "v": "<value>" }**. The JSON message created in step 14 would be **{ "n": "temperature", "v": 35.0 }**. Now let's look at the code again. Line 42 is how a

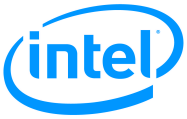
JSON message is created. The variable “component” is the second command-line argument (e.g. **temperature** in step 14). The variable “value” is the third command-line argument (e.g. **35.0** in step 14). The first argument is the program file itself.

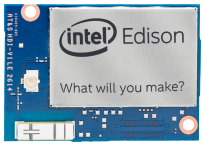
Figure 21 The Arguments

Upon the receipt of the message, the IoT agent add a security token, add a time stamp, convert "temperature" to the component (time series) ID, and send a POST over SSL to the cloud server [5].

Optional Practice

1. Write a C code that works the same as “**send_udp.js**”.
2. Try to write an alternative version using TCP protocol instead of UDP protocol.





Custom Components

So far, we have only sent observation data specified by us. “iotkit-admin observation temperature 30” or “./send_udp.js temperature 30” sends the temperature observation data of 30 degrees Celsius that is not a real sensor measurement. In this section, we will use a rotary sensor from Grove Starter Kit to demonstrate not only how to collect real sensor data but also how to register a custom component.

1. First, let’s register a rotary sensor as a custom component.
2. A rotary sensor is neither a humidity sensor nor a temperature sensor, so we need to add a new item to the Catalog.
3. On the dashboard, go to **Catalog** tab of **My Account** page.
4. Scroll down to the bottom of the page and click **Add a New Catalog Item**.
5. Fill in the fields as shown below.

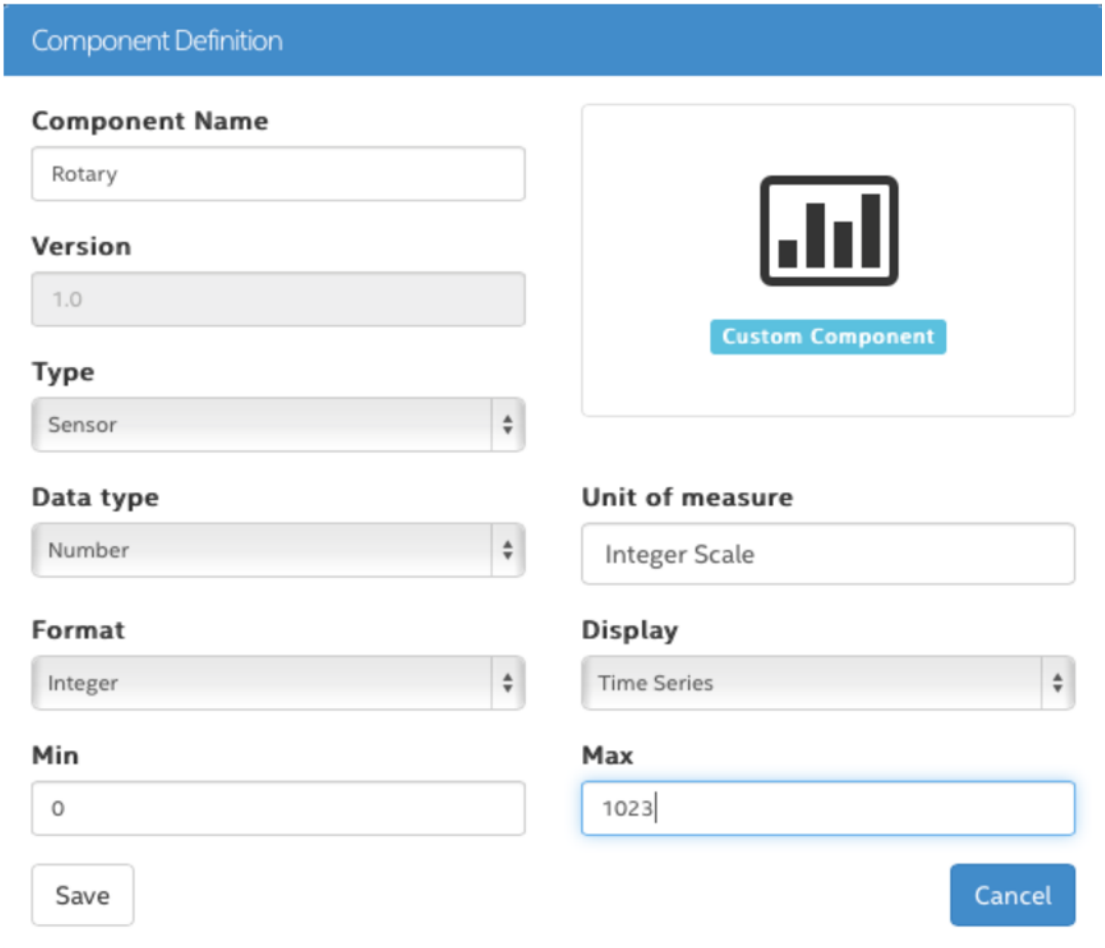


Figure 22 Component Definition

6. Click on **Save**.
7. Serial connect or SSH into your Edison.

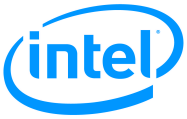
8. **$ iotkit-admin catalog**

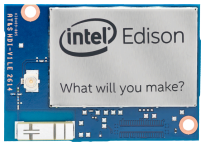
```
root@chris_edison:~/tutorial4_examples# iotkit-admin catalog
2015-07-08T02:33:07.576Z - info: Starting Catalog Retrieving
2015-07-08T02:33:08.431Z - info: Comp:  rotary.v1.0   Rotary   sensor
2015-07-08T02:33:08.435Z - info: Comp:  button.v1.0   button   sensor
2015-07-08T02:33:08.436Z - info: Comp:  humidity.v1.0   humidity   sensor
2015-07-08T02:33:08.437Z - info: Comp:  powerswitch.v1.0   powerswitch   actuator
2015-07-08T02:33:08.438Z - info: Comp:  temperature.v1.0   temperature   sensor
2015-07-08T02:33:08.442Z - info: # of Component @ Catalog :  5
```

id : t	kind	measure
rotary.v1.0	sensor	Rotary
button.v1.0	sensor	button
humidity.v1.0	sensor	humidity
powerswitch.v1.0	actuator	powerswitch
temperature.v1.0	sensor	temperature

Figure 23 Catalog Output

9. The output indicates that we have a new catalog item with ID "**rotary.v1.0**".
10. **$ iotkit-admin register rotary rotary.v1.0**
11. As mentioned above, the IoT agent needs to be restarted in order to update the new component registration.
12. **$ systemctl status iotkit-agent**
13. If you are not in "/home/root/cloud_analytics" directory already, navigate to the directory.
14. **$ vi cloud_rotary.c**
15. Type the following C code.






```
#include <stdio.h>
#include <unistd.h>
#include <mraa/aio.h>

int main()
{
      uint16_t value = 0;
      char cmd_buf[1024];
      mraa_aio_context rotary;
      rotary = mraa_aio_init(0);

      while(1){
            value = mraa_aio_read(rotary);
            snprintf(cmd_buf, sizeof(cmd_buf), “./send_udp.js rotary
%d", value);
            system(cmd_buf);
            printf("%d\n", value);
            sleep(1);
      }

      mraa_aio_close(rotary);
      return 0;
}
```

Figure 24 cloud_rotary.c

16. **$ gcc -lmraa -o cloud_rotary cloud_rotary.c**
17. **$ ./cloud_rotary**
18. Turn the sensor slowly for a few seconds.
19. Press **Ctrl-C** to quit.
20. Go to **My Charts** page on the dashboard and select rotary component.
21. Compare the result with the output on the terminal.

Optional Practice

- Modify rotary.c so that it directly sends a UDP-formatted message to the IoT agent rather than making a system call to run “**send_udp.js**”.