Master your IoT development skills by extending an IoT system

Enhance your IoT system with device management and also with analytics by applying rules and actions to your IoT data

Anna Gerber

January 03, 2018
(First published December 22, 2017)

In this tutorial, master your IoT development skills by extending an existing IoT system. In just 3 steps you explore the device management, visualization, and analytics capabilities of Watson IoT Platform.

IoT 301: Mastering IoT development

This article is part of the IoT 301 learning path, an advanced developer guide for IoT.

- Get serious
- IoT security challenges
- IoT device management
- IoT analytics
- Tutorial: Extend an IoT system (this tutorial)

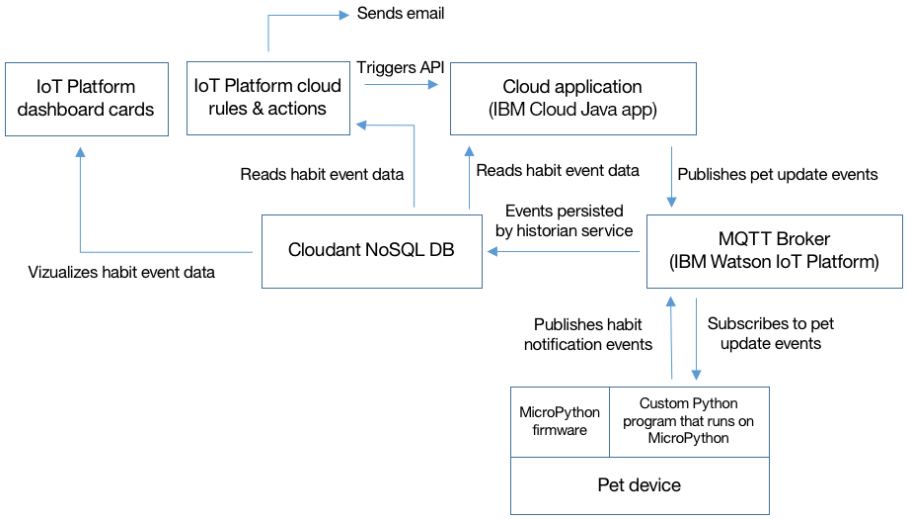
In my tutorial "Build your skills in IoT development by developing a Healthy Habits Tracker," I showed how to build a habit tracking IoT device, my Healthy Habits Pet, which encourages me to take regular breaks from my computer for exercise. The healthy habits pet sends and receives data by using MQTT. It receives messages from a cloud application that periodically trigger notifications that activate the device's LEDs and piezo buzzer. And, it publishes MQTT events whenever I tap the pet, to indicate that I have completed my exercise.

This tutorial builds on that project and walks through how I extended the device's MicroPython program to enable the device to be managed by the IoT platform, how I set up basic visualizations using device data from within the IBM Watson IoT Platform dashboard, and how I configured rules to trigger actions to motivate me to continue with my exercise habit. Watch the following video where I introduce this project.

To view this video, **Healthy Habits Tracker - Introduction** , please access the online version of the article. If this article is in the developerWorks archives, the video is no longer accessible.

The architecture for the extended application is shown in Figure 1.

Figure 1. Architecture of the extended Health Habits Tracker application



What you'll need

You need all of the prerequisite hardware, programs, and accounts for my previous tutorial, Build your skills in IoT development by developing a Healthy Habits Tracker. And, of course, you'll need to develop your own healthy habits pet by completing that tutorial.

1. Extending the device's MicroPython program

The updated device progam can be found in the managed-healthy-habits-pet git repository.

Get the code

I extended the device program by adding an additional property, adding managed device support, and adding deep sleep support.

1a. Adding a property

I added an additional property to the data payload that is being sent by the device over MQTT to the Watson IoT Platform. The property is used to indicate how energetic I'm feeling, and this is determined based on how long I hold down the button when recording my exercise. If I tap the device quickly, I'm recording that I'm tired, while if I hold the button down for longer, I'm indicating that I'm full of energy. This is a simple way to record my energy level without interrupting my regular routine or having to add any additional sensors to the device. However, the results aren't precise – I'm simply sending through the number of milliseconds that the button is held down as the raw data for this property, so it's only useful as a rough indicator.

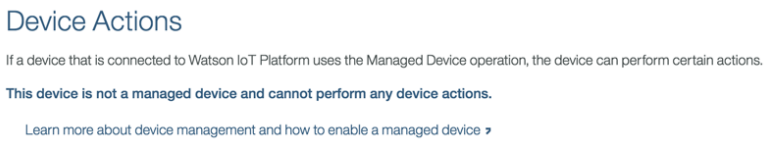
I'm using the utime module that comes built into MicroPython to get the number of milliseconds when the button-down event is detected, then calculating the diff of when the button is released, and then adding that result to the payload that is later sent over MQTT:

```
if firstButtonReading and not secondButtonReading:
  buttonPressStarted = utime.ticks_ms()
elif not firstButtonReading and secondButtonReading:
  duration = utime.ticks_diff(utime.ticks_ms(), buttonPressStarted)
  payload['energy'] = duration
```

1b. Adding managed device support

You can use the IBM Watson IoT Platform device management protocol to configure your IoT devices as managed devices, which allows the devices to be rebooted, to be reset to factory defaults, or for firmware to be downloaded or upgraded remotely.

Even though I had previously registered my healthy habit pet device with the Watson IoT Platform, it was not a managed device, so at the bottom of the device detail page, no device actions were available.



Managed devices implement the Device Management Protocol, which is built on MQTT.

To configure my healthy habits pet as a managed device, the device needs to publish an MQTT message (the managed data request) to the `iotdevice-1/mgmt/manage` topic with information about the actions it supports. These can be firmware actions (for updating and downloading the device program, or firmware) or device actions (for rebooting the device and updating device settings). Let's extend the MicroPython program for my healthy habits pet device to include support for device actions by publishing the data to the manage topic:

```
deviceData = {'d': {'lifetime': 0, 'supports': {'deviceActions': True}}}
client.publish(b"iotdevice-1/mgmt/manage", ujson.dumps(deviceData))
```

The `supports` property lists the types of operations that are supported (in this case, only deviceActions is supported), and the `lifetime` property in the payload indicates how many seconds may pass before the device is considered to be dormant. If it has not sent an updated managed devices message, the 0 value means that the device will never be considered dormant.

The device must also subscribe to the `iotdm-1/device/update` topic to receive messages to trigger these actions, and implement the functionality. The device can also subscribe to the `iotdm-1/response` topic to receive the responses sent by Watson IoT Platform in response to managed data requests. Use the client.subscribe function in MicroPython to subscribe to these topics:

```
client.subscribe(b"iotdm-1/response")
client.subscribe(b"iotdm-1/device/update")
```

I implemented two device management functions. I defined a reboot function by using the MicroPython machine.reset function to be able to reboot my healthy habits pet device, and I defined a factory reset function by resetting the status of my healthy habits pet device to good.

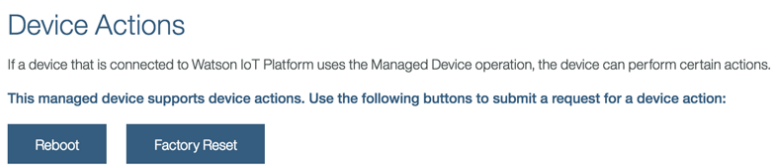
```
def rebootDevice():
  # trigger device to hard reset
  machine.reset()

def factoryReset():
  # reset device to 'good'
  global currentStatus
  currentStatus = 'good'
  updateEyes(currentStatus)
```

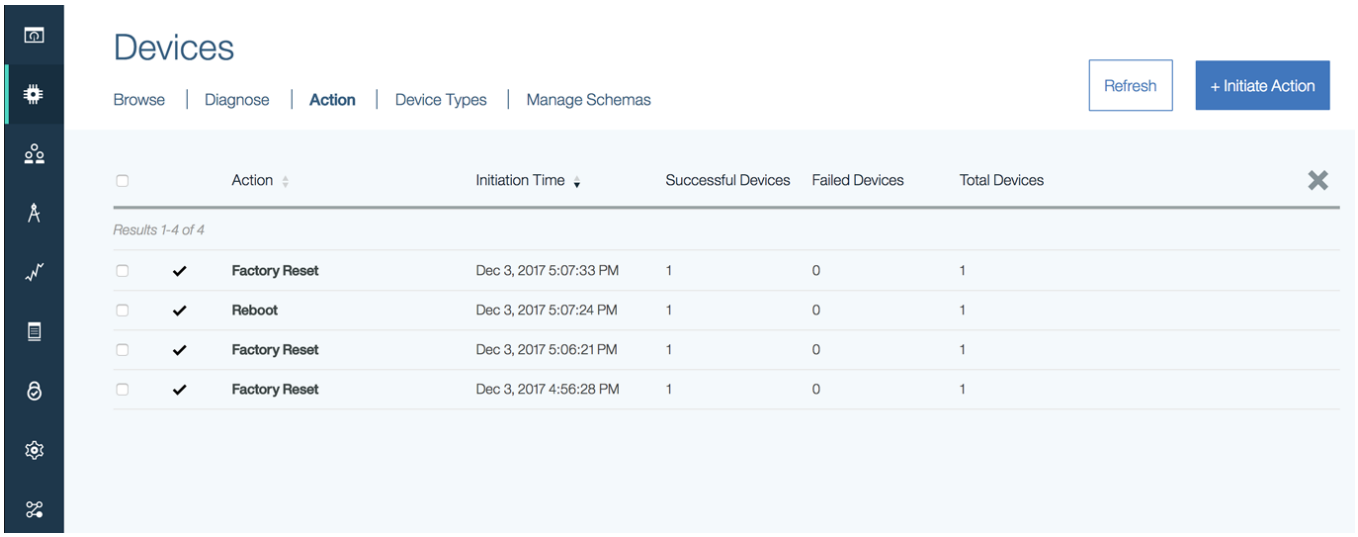
Follow along in this video as I show you how to add device actions to managed devices.

To view this video, **Extended Healthy Habits Tracker - Applying actions to managed devices** , please access the online version of the article. If this article is in the developerWorks archives, the video is no longer accessible.

After registering my healthy habits pet device as a managed device by sending the MQTT message, the Device Actions section on the device detail page in the Watson IoT Platform dashboard is automatically updated to include buttons that can be used to trigger these device actions.



In addition, a log of device actions that have been applied is available under the Action tab on the Devices page in the dashboard, and the Initiate Action button on this page can also be used here to trigger actions. These actions can be applied to a single device or to all devices of a given type at the same time.



1c. Implementing deep sleep

One of my motivations for selecting the AdaFruit Feather Huzzah ESP8266 microcontroller for this project was that the board includes a LiPo connector and charging circuitry on board, so my device

can be powered off battery. However, with the device constantly connecting and checking for new MQTT messages, the healthy habit pet will not last long on battery power.

From looking at the frequency and timestamps on the habit events that were stored in the Cloudant database, I can see that I typically only exercise once every few hours. So, it makes sense from a power management point of view to sleep the device immediately after performing exercise. For this to work, I connected a wire from GPIO 16 to the RST pin on the microcontroller. (Review the videos from Step 1 in the previous tutorial to see how to add this wire.)

Sleeping the device from MicroPython involves setting up a real-time clock alarm (which will wake the device after a certain period), and then calling the deepsleep function:

rtc = machine.RTC()

rtc.irq(trigger=rtc.ALARM0, wake=machine.DEEPSLEEP)

```
# sleep for twenty minutes
rtc.alarm(rtc.ALARM0, 1200000)
machine.deepsleep()
```

I'm calling this immediately after sending an exercise event, but it could also be triggered in response to MQTT messages, as a way to implement remote power management.

1d. Adding device, data, and application security

When working with IoT data, the data is only as secure as the devices and apps that produce and consume it.

> My Top 10 IoT security Challenges blog post outlines some of the key considerations for IoT security. You can also learn more about securing IoT data in Part 2 of the Design and build secure IoT solutions series.

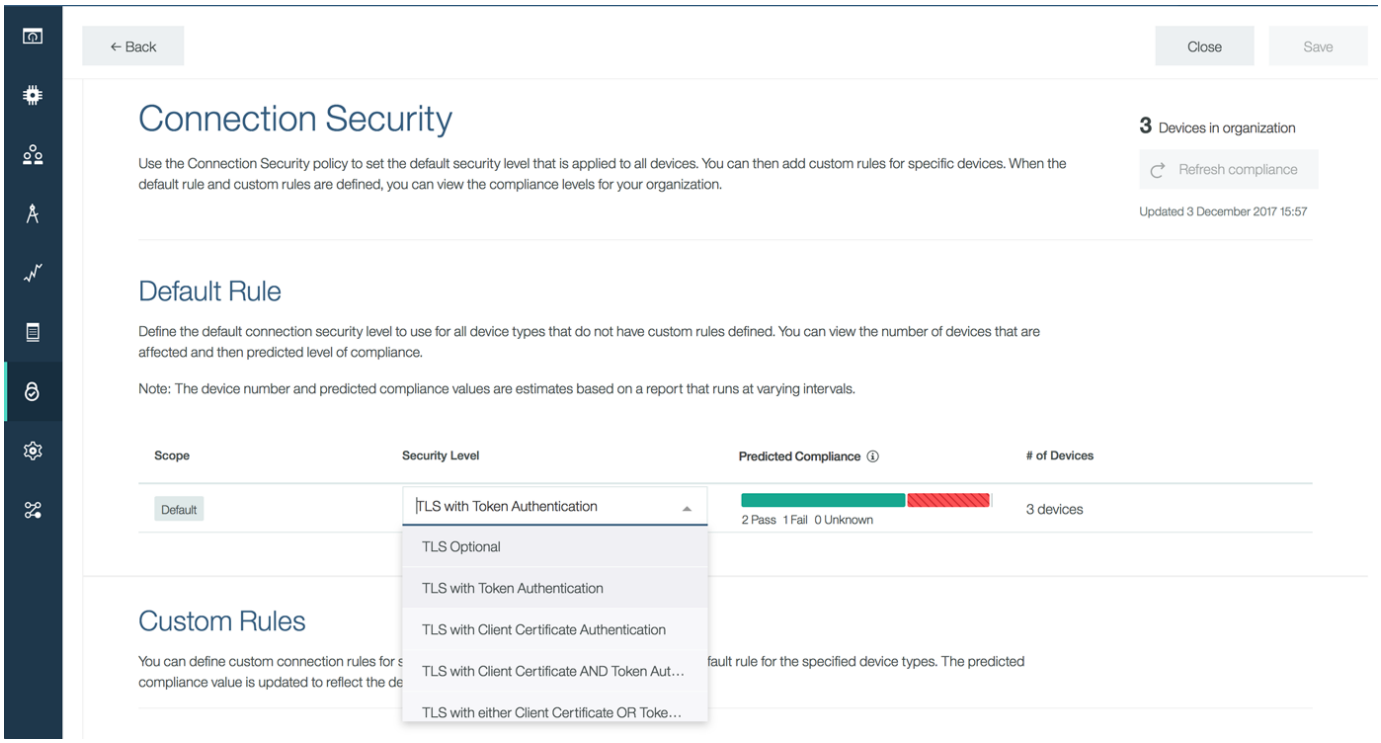
The AdaFruit Feather Huzzah ESP8266 is a constrained device, so it is recommended that you ensure that the healthy habit pet devices are partitioned from other parts of the network and that you use firewalls when deploying the devices.

Device authorization and authentication helps to protect the data that is being stored by guaranteeing that the data being stored was generated by a known device. Using the IBM Watson IoT Platform device manager means that this is taken care of in this application; when I registered the device, an authentication token was generated that the device uses for all MQTT communication. I can revoke the device's access by deleting the device from within the devices section of the Watson IoT Platform dashboard (which removes the token). However, be careful when resetting devices to factory settings, because if the token is lost, the device will need to be re-registered (a token cannot be retrieved).

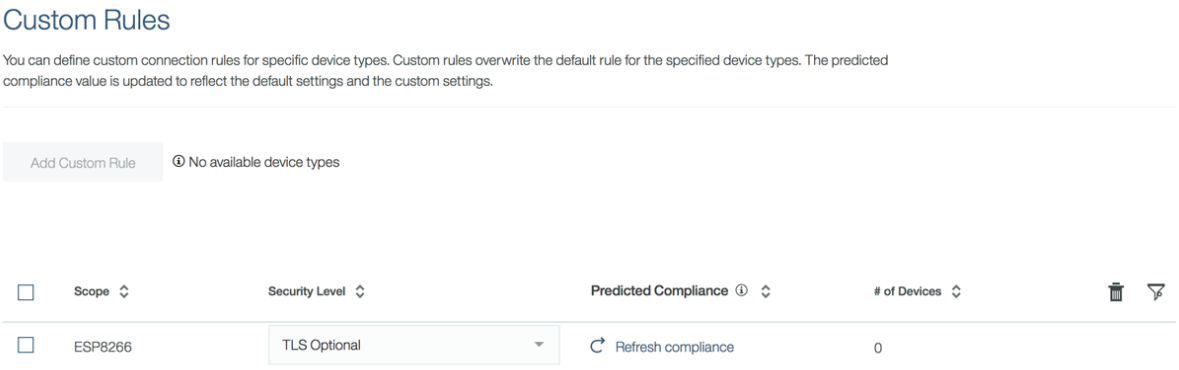
Secure communication is also important for ensuring data privacy. It is preferred to use TLS or SSL when connecting the MQTT client to the platform. However, there is currently a known limitation with checking for new MQTT messages in a non-blocking way using MicroPython on ESP8266 devices over TLS connections, which means that the button press events are not able

to be sent while the device is waiting to receive subscribed messages. To work around the issue, my device needs to connect to MQTT without TLS to be able to subscribe to and receive MQTT events while also being able to poll the button state to detect button press events and trigger sending the data over MQTT.

The connection security settings can be accessed from the Security menu item within the Watson IoT Platform dashboard. The default rule is to use TLS with token authentication; however, there is also an option to require client certificate authentication (or both), as well as being able to configure rules for specific device types.



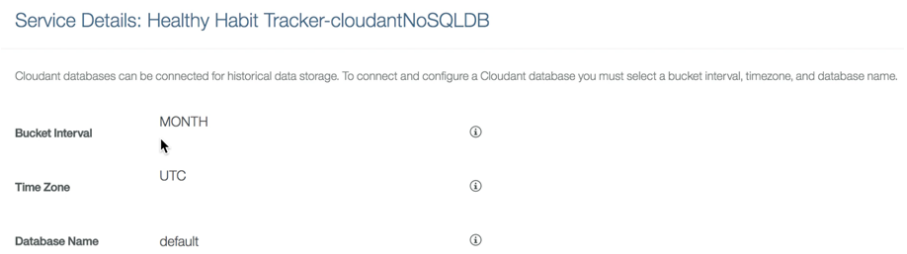
I've set up one of these custom rules to make TLS optional for my ESP8266 device type only. This makes it possible for my device to both send and receive MQTT devices, given the current constraints of the device.



The Java application also needs to be secured. The application connects securely over TLS to the MQTT broker to publish reminder notifications, and uses an API key and token auth to authenticate with the Watson IoT Platform. The application also uses a credentials URL to authenticate with the Cloudant NoSQL database. Because the data in this application is not particularly sensitive, I have not implemented user authentication, however it could be added using Java Authentication and Authorization Services (JAAS) framework or the IBM Cloud App ID service to authenticate using existing social logins.

2. Visualizing device data

In a previous article in this learning path, "Making sense of IoT data," I describe some of the tools and approaches for analyzing IoT data. For this application, I'm using the analytics capabilities that are built in to the IBM Watson IoT Platform. The data that the healthy habits pet device sends to the platform over MQTT is stored in a Cloudant NoSQL database using the Historical Data Storage extension, bucketed by month:



Dashboard cards can use the data as a source for visualizations including latest values, line charts, bar charts, or gauges. It's also possible to create custom cards to support other kinds of visualizations.

The following video shows how the device data is persisted to cloud storage and how to set up a visualization card using data within the Watson IoT Platform dashboard as well as how to configure a device schema.

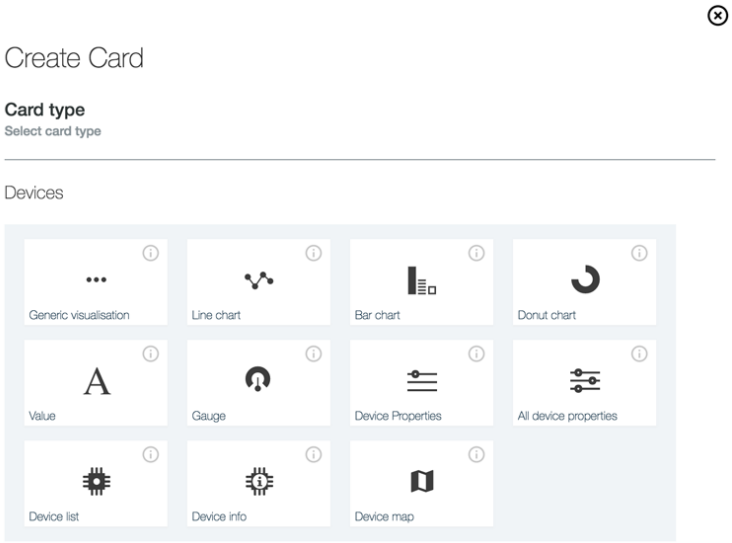
To view this video, **Extended Healthy Habits Tracker - Visualizing your data** , please access the online version of the article. If this article is in the developerWorks archives, the video is no longer accessible.

2a. Add a visualization card

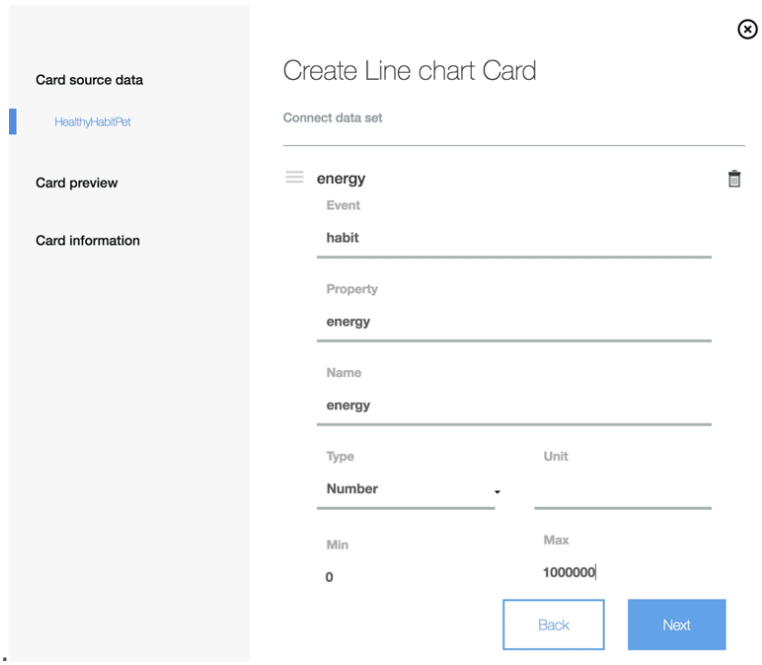
Device data that is stored within one or more Cloudant NoSQL databases can be visualized or displayed on dashboard boards within cards. To configure a card to visualize device data, select the board where the card should appear, and then:

1. Click **Add new card**.

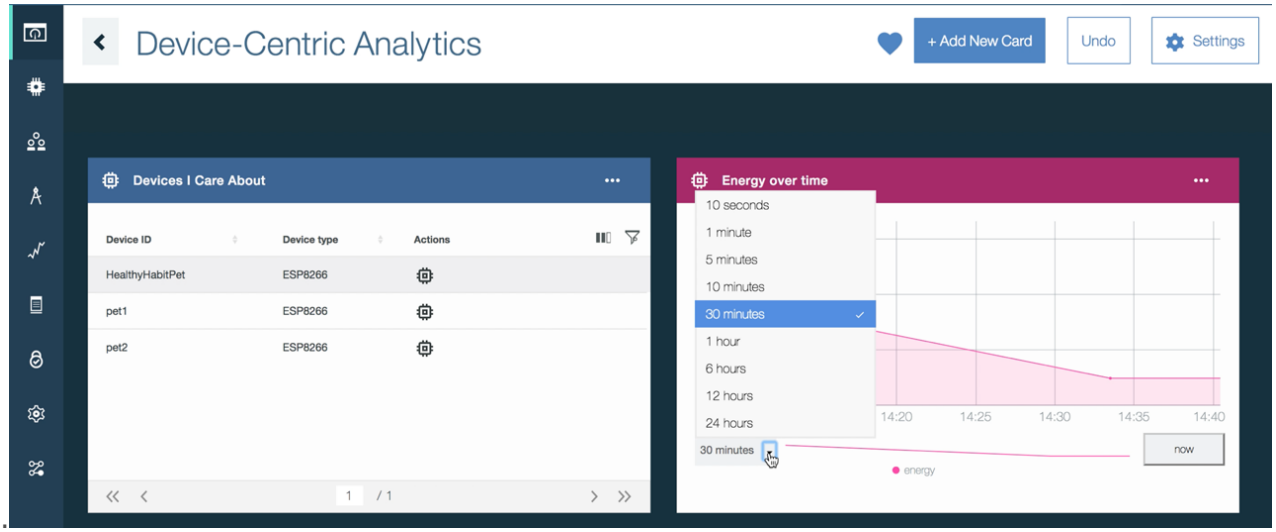
2. Select the card type from the options available, for example, select **Line chart** under **Devices**.



3. Select the device that will provide the data to be displayed in the card. Some card types provide an option to use another card as a data source for the data set that is displayed. This allows the data displayed on the dependent card to be filtered by selecting values on the other card, for example, the values that are included in a visualization card can be filtered by selecting which devices to display from a device list card.
4. Connect a data set. If you haven't configured a device schema, you'll need to select the event type (habit), and then configure a property by selecting the data type and entering a range of values and then clicking **Next**. Or, if you have configured a device schema, you can select a property from the schema to save time.



5. Configure the appearance of the card, by entering a size, color, and title for the card, and then click **Submit** to add the card to the board.

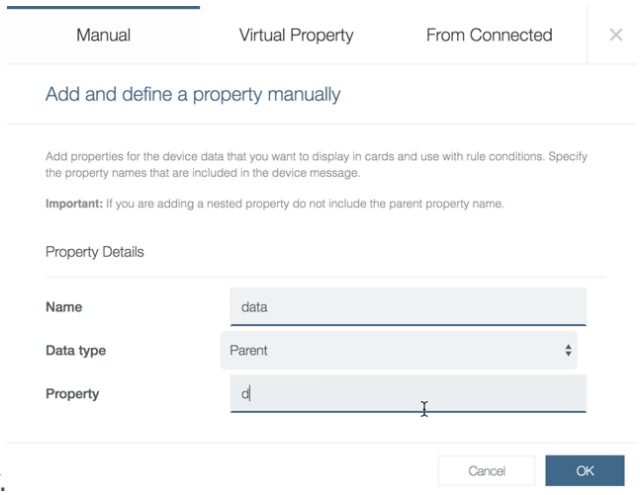


2b. Create a device schema

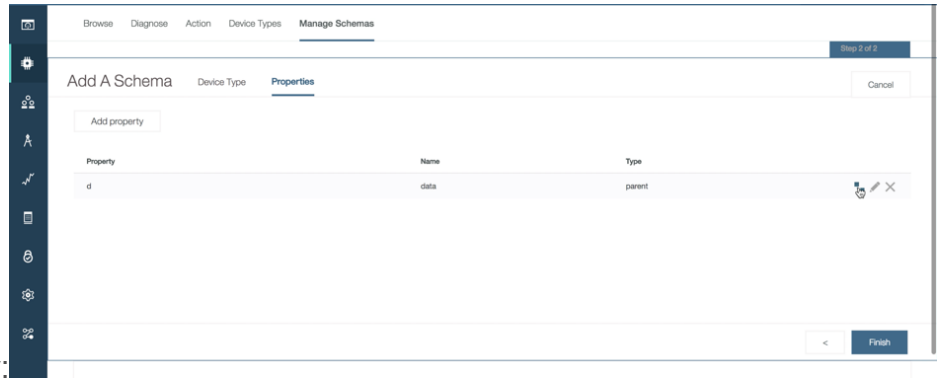
The platform supports configuring a device schema to describe the properties that are available for use within analytics or for use in configuring rules and actions.

To configure a device schema, follow these steps:

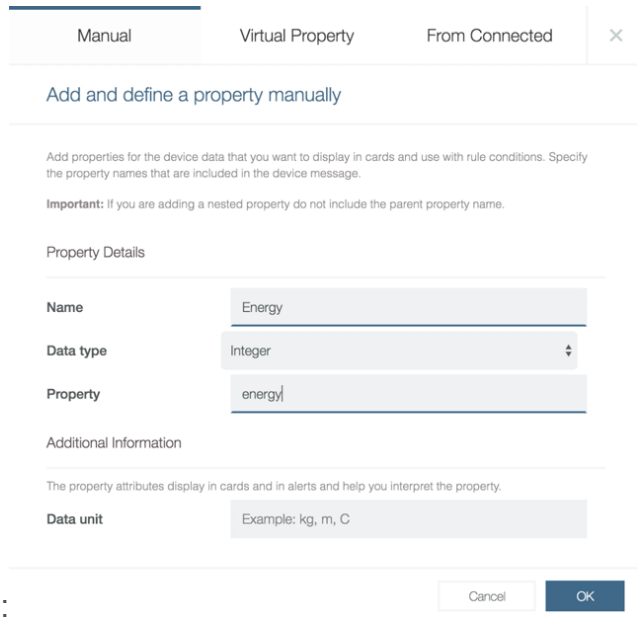
1. Select **Devices** from the left-hand sidebar, and then select **Manage schemas**.
2. Click **Create a schema**, and then select the device type (in this case, ESP8266).
3. Click the **Add a property** link, and add a property that corresponds to the `d` property from the device JSON payload, with the property data type of `Parent`. You will need to enter the `Name` field for the property, which is a human readable name that will help you to identify the property, for example I used the name `data` as a human readable name for the `d` property.



4. Click the **create child** button shown at the far right of the `d` property to add a new child property:



5. Enter the property details for the `energy` property, by specifying a human readable name (`Energy`), selecting the `Integer` data type and entering the corresponding property from the device JSON (`energy`):



6. Once you have saved the schema, the properties that are defined in the schema will be available for use within card visualizations and rules.

3. Setting up rules and actions

> You can read more about additional actions that are supported, such as sending an email, triggering a Node-RED process, and integrating with IFTTT, in the "Perform Actions in IBM Watson IoT Platform Cloud Analytics" developerWorks recipe.

Previously, my healthy habit pet displayed behaviors based only on simple triggers. It played a happy sound effect and displayed animated hearts in the eyes immediately after I pressed the button after completing some exercise, and it displayed different eye animations in response to the periodic reminders that are generated within my Java-based cloud application. I've set up some rules and actions to expand this behavior.

I extended the REST API in the cloud application (available in my GitHub repository) so that it takes a parameter for the status to send as part of the reminder, rather than a hardcoded 'good' status as it was before:

```
@GET
     @Path("/remind/{status}")
     public String remind(@PathParam("status") String status) {
   reminderPublisher.remind(status);
   return "OK";
     }
```

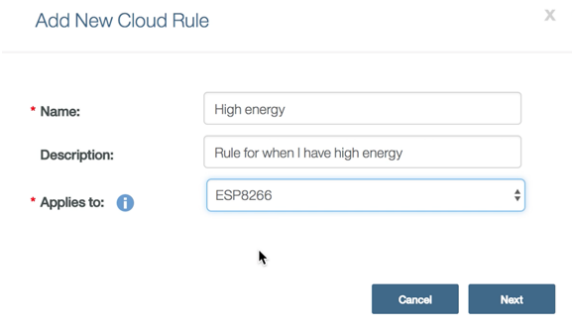
I'm using this API endpoint as a web hook so I can trigger different types of notifications (sleepy, happy, and so on) based on conditions that are triggered through rules, such as when my energy property drops below a certain threshold.

The following video shows how to set up rules to trigger a web hook and send an email.

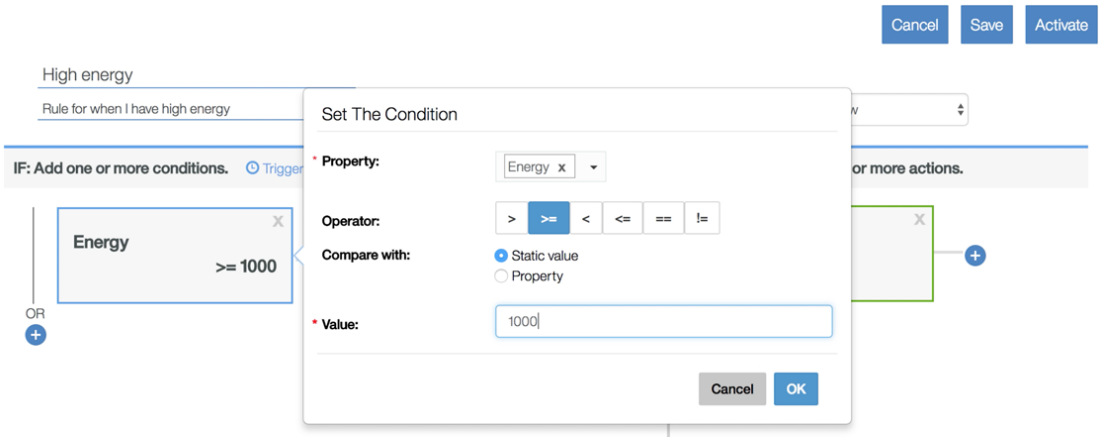
To view this video, **Extended Healthy Habits Tracker - Creating rules** , please access the online version of the article. If this article is in the developerWorks archives, the video is no longer accessible.

To set up a rule, select 'Rules' from the Watson IoT Platform dashboard sidebar, and then click on the **Create Cloud Rule** button to set up a new rule.

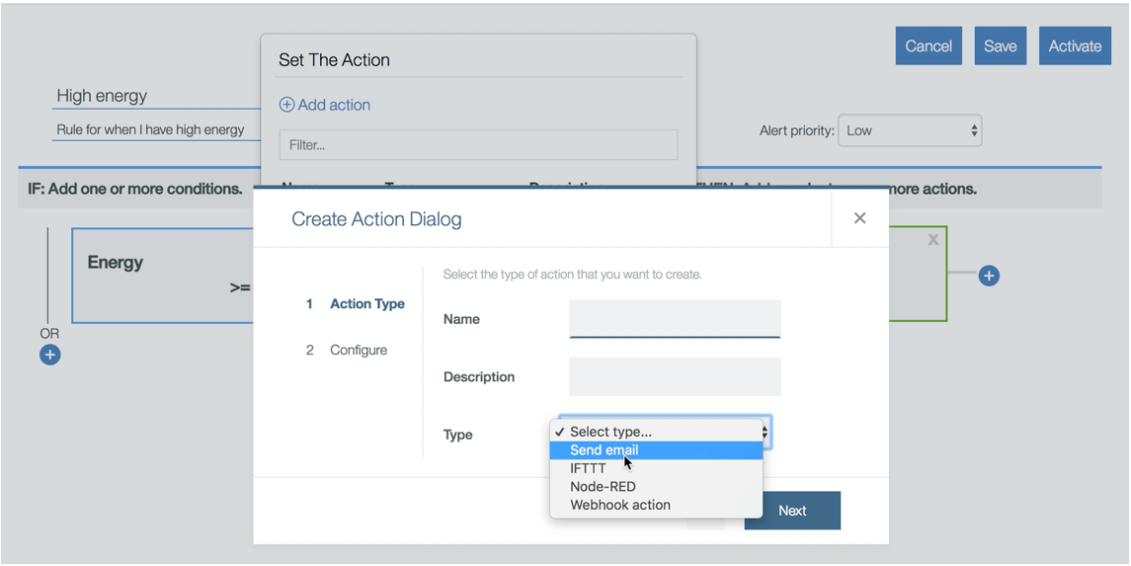
1. Enter a name, description, and device type for the new rule:



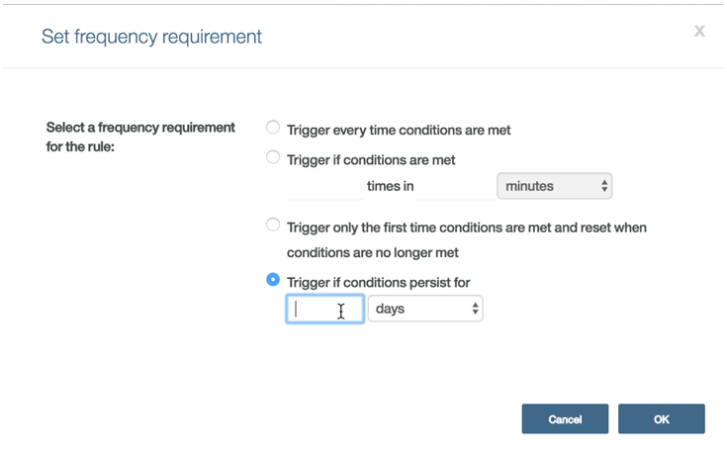
2. Add one or more conditions on the left panel of the rule editor. For example, I have configured this rule to trigger when the value of the energy property is greater than or equal to 1000.



3. An alert will be generated when the rule conditions are matched. The priority of the alert (for example, Low) can be set using the drop-down menu in the editor. Actions can be configured to be applied whenever the rule is activated.
4. To add an action, click the **New action** button and then select an existing action, or add a new action by choosing between sending an email, integrating with IFTTT, triggering a Node-RED process, or triggering a Web hook action. Finally, follow the prompts to set up the action.



5. By default, the rule will be set up to trigger every time conditions are met. However, you can customize the frequency requirement for the rule by clicking on the trigger and selecting from the frequency options available, as shown in the following figure:



Summary and wrap-up

I hope you enjoyed extending and enhancing the healthy habits tracker and learned even more about the breadth of features that an IoT Platform (in this case IBM Watson IoT Platform) can bring to your IoT solutions. From device management, to data visualization, to triggering actions based on rules you define, you learned (perhaps even mastered?) some of the more advanced IoT development skills.

Related topics

- IoT 201: Build your skills in IoT development by developing a Healthy Habits Tracker
- Watson IoT Platform Documentation for visualizing real-time data
- Watson IoT Platform Documentation for creating rules and actions
- IoT articles and tutorials on developerWorks
- Hands-on IoT videos on developerWorks TV
- All IoT videos on developerWorks TV
- Community-contributed tutorials on developerWorks Recipes
- IoT courses for developers