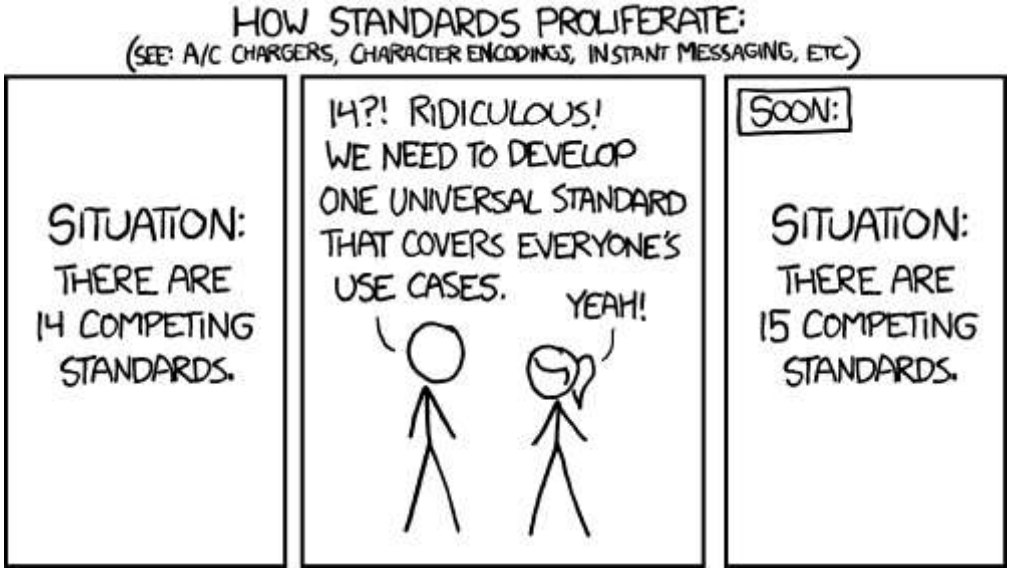
Transform a RaspberryPi into a universal Zigbee and Z-Wave bridge

Automation Guru Follow
Feb 25 · 10 min read ★

Home automation comes with plenty of potential to make our lives easier. But in order to succeed in its task, it often requires you to fill your house with bridges that can connect your smart devices to your Wi-Fi network. Unless you buy a smart device that communicates directly over Wi-Fi (like a TP-Link or Belkin smart plug), odds are that many of your favourite smart devices use either Bluetooth, Zigbee or Z-Wave to communicate. These protocols solve some of the issues of Wi-Fi when it comes to smart

devices — like latency, centralised topology and relatively high power requirements — but they do require some physical hardware in between to do the smart protocol <-> Wi-Fi translation and make the devices actually controllable from a Wi-Fi-connected client. The bad thing is that you'll probably need a different bridge or hub for each class of devices you want to use. Philips Hue lights come with their own bridge, same for Lutron, same for HomeKit, same for Belkin, same for Switchbot, and the list goes on. What's ironic is that most of these devices actually speak the same protocol (either Zigbee or Z-Wave) but, in most of the cases, they can only control their own devices. Try to imagine an alternative reality where all the ethernet cards can send and receive TCP/IP packets, but you'll need an adapter just for HTTP traffic, one for FTP, one for SMTP, and so on: that's more or less the reality today when it comes to smart bridges and hubs. In order to solve the problem, many companies are throwing on the market even *more* hubs and bridges (from Samsung SmartThings, to the plethora of Amazon Echo, Google and Apple devices etc.), often compatible only with a subset of the devices out there and incompatible with one another, often incompatible by design with devices produced by competitors: that only brings even more fragmentation to the current smart home hell.



Moreover, many hubs can usually talk only to their smartphone app, adding an app hell to the bridge hell.

One of the most sought goals nowadays when it comes to home automation is (or it should be) to find consistent ways to communicate with as many devices as possible using the same tool, the same box and the same interface.

I've recently worked on achieving this goal in platypush. Other platforms have tackled the issue too (from HASS.io to Openhab) but, in my opinion, they all still require a certain degree of user configuration (like specifying what each device is and what it's supposed to do) that in my opinion shouldn't be required. In this article, we'll go through what Zigbee and Z-Wave actually are, what you need to set up a universal bridge for them using just a Raspberry Pi and a USB dongle, and how to permanently move your existing smart bridges to the storage room.

Zigbee vs. Z-Wave

Zigbee and Z-Wave have arguably risen to become the dominant communication protocols in the smart home and IoT industries. They share some common features, such as:

- They are better suited for low-powered devices. Wi-Fi is a relatively power-hungry protocol, needs to deal with packet losses, relies on a centralised topology, and isn't ideal for communication with devices that can be asleep for a long time because of its high overhead on reconnection. Zigbee and Z-Wave have instead been designed to work well with low-powered devices, be low-latency and optimized for sending less bytes on the network. Some Z-Wave or Zigbee devices can theoretically run on the same battery for a couple of years; a similar Wi-Fi device won't usually last more than a couple of days.

- While Wi-Fi networks are usually configured using a star-based topology (one access point/router in the middle and all the devices connect to it), Zigbee and Z-Wave have been designed to support more flexible topologies. Each device can be both a client and a repeater on the network, each can directly connect to other devices and expand the network with no need to introduce further routers or access points. This allows the creation of scalable *mesh* networks that can include devices that would otherwise be out of range.

When it comes to the differences:

- **Number of devices**: Z-Wave networks are limited to 232 devices, while Zigbee supports up to 65k devices.

- **Maximum hops**: Z-Wave supports up to four hops between a device and the network controller, while Zigbee has theoretically no such limitations.

- **Operating range**: Most Zigwave devices operate around the 2.4 GHz spectrum — although some specific devices also use 784 MHz in China, 868 MHz in Europe and 915 MHz in the US and Australia. Z-Wave devices instead operate around the 850–900 MHz range (at 868.42 MHz in Europe, at 908.42 MHz in North America and other frequencies in other countries depending on their regulations).

- **Signal range**: A thumb rule when it comes to electromagnetic waves is that, at fixed transmission power, the higher the frequency the lower the range. It means that Z-Wave devices, which operate at lower frequencies, have usually a longer range compared to Zigbee. An unobstructed Z-Wave signal can travel up to 100 meters outdoor, although a common guideline is 30 meters indoor for unobstructed signals and 15 meters if there are walls in between. The indoor range for Zigbee is instead usually limited around 12 meters. However, Zigbee networks theoretically support an unlimited number of hops between the nodes and the controller, so the range can be easily extended by adding more nodes to the network.

- **Signal reliability**: Z-Wave has a longer range than Zigbee. It means that, at a fixed distance, its signal is usually more reliable. Moreover, Zigbee operates in the relatively "quieter" 800–900 MHz spectrum, therefore it doesn't have to share the crowded 2.4 GHz spectrum with Wi-Fi and Bluetooth devices. That overall leads to more reliable communication.

- **Ownership and protocol**: Z-Wave is a proprietary technology developed and maintained by Sigma Designs. Sigma (acquired by Silicon Labs in 2018) owns the protocol, licenses the compatible devices and runs the Z-Wave Alliance, and grants certifications to the devices that comply with the standard. Its selling point is, in my opinion, the strong enforcement of a shared protocol both on hardware and software side. Each sensor, configuration value or switch is defined by a variable type, a range, a read-only vs. read-write policy and a structured representation that applies to all the compliant devices. That makes it very easy to develop consistent interfaces that can comprehensively represent and control any device as long as it speaks Z-Wave, even if the implementation doesn't know exactly which device it is. Zigbee, on the other hand, is an open standard maintained by the Zigbee Alliance. It

also has a certification process in place, but that comes in two parts — one part certifies the hardware, the other certifies the software. It is possible to produce Zigbee certified hardware even if the software isn't certified nor compatible with other clients. While this makes the protocol much more open than Z-Wave, it also makes the task of writing an all-purpose interfacing software much harder, as different devices may name their properties following different conventions.

That should cover most of the knowledge you need when it comes to the theory of Zigbee and Z-Wave. You'll find many Zigbee and Z-Wave compatible smart devices around. Some examples of Zigbee devices are the Philips Hue and Ikea smart bulbs, Honeywell thermostats, Belkin smart plugs and bulbs, Bosch sensors and Osram products. Z-Wave includes around 65,000 compatible devices out there, including many garage doors, presence and temperature sensors, thermostats, dimmers, remote controls, smoke detectors and so on.

Hardware and software

We'll use a RaspberryPi in the following examples as a DIY bridge (any model and any distribution should work fine), and platypush as a home automation platform that also runs the Zigbee and Z-Wave integrations.

Making your own Zigbee bridge

On the hardware side you'll need:

- One or more Zigbee compatible devices, like Philips Hue, Ikea or Osram lights or Belkin switches.
- A Zigbee-to-USB adapter/sniffer. The CC2531 is one of the most popular options out there.
- A Zigbee debugger+adapter cable, that you'll need in order to flash the firmware on the dongle.
- A RaspberryPi or similar device (any model and distribution should work fine), or any computer that you want to use as a bridge.

On the software side:

- platypush uses zigbee2mqtt as a backend to interact with the Zigbee dongle. Install the zigbee2mqtt firmware on the dongle by following the instructions (for Windows, macOS and Linux) on their website. You can check a list of the compatible devices here.

- Install, start and enable an MQTT instance on the local machine, if you don't have a server already running in your network. If you're running Debian/Ubuntu/Raspbian and want to use Mosquitto, for example:

```
[sudo] apt-get install mosquitto
[sudo] systemctl start mosquitto.service
[sudo] systemctl enable mosquitto.service
```

- Install zigbee2mqtt:

```
[sudo] apt-get install nodejs git make g++ gcc
git clone https://github.com/Koenkk/zigbee2mqtt
cd zigbee2mqtt
npm install
vi data/configuration
# Change mqtt.server and serial.port to respectively
# match your MQTT server and USB dongle device.

npm start
```

You can also make a systemd service out of it:

```
[Unit]
Description=zigbee2mqtt
After=network.target

[Service]
ExecStart=/usr/bin/npm start
WorkingDirectory=/path/to/zigbee2mqtt
StandardOutput=inherit
StandardError=inherit
Restart=always

[Install]
WantedBy=multi-user.target
```

- Note that the zigbee2mqtt configuration file also includes a `permit_join` option. Set this to true while you're pairing your Zigbee devices for the first time, and set it to false afterwards to prevent other devices from accidentally or malignantly join the network — you can always temporarily allow joins later.
- Once started and in `permit_join` mode, you can start pairing Zigbee devices to your new network. That is usually done by doing a factory reset of the device. The procedure varies with the device: Philips Hue bulbs, for example, can be reset either from the app (if they are paired to a bridge) or by pressing the ON/OFF buttons of a Hue dimmer simultaneously for 10 seconds while keeping the dimmer within 10 cm from the lightbulb. Other Zigbee devices may include a reset button instead.
- Once a Zigbee device joins the network, the zigbee2mqtt logs should show a trace like the following:

```
Successfully interviewed '0x00158d0001dc126a', device has
successfully been paired
```

- Install Redis and platypush with the Zigbee, HTTP and MQTT extensions:

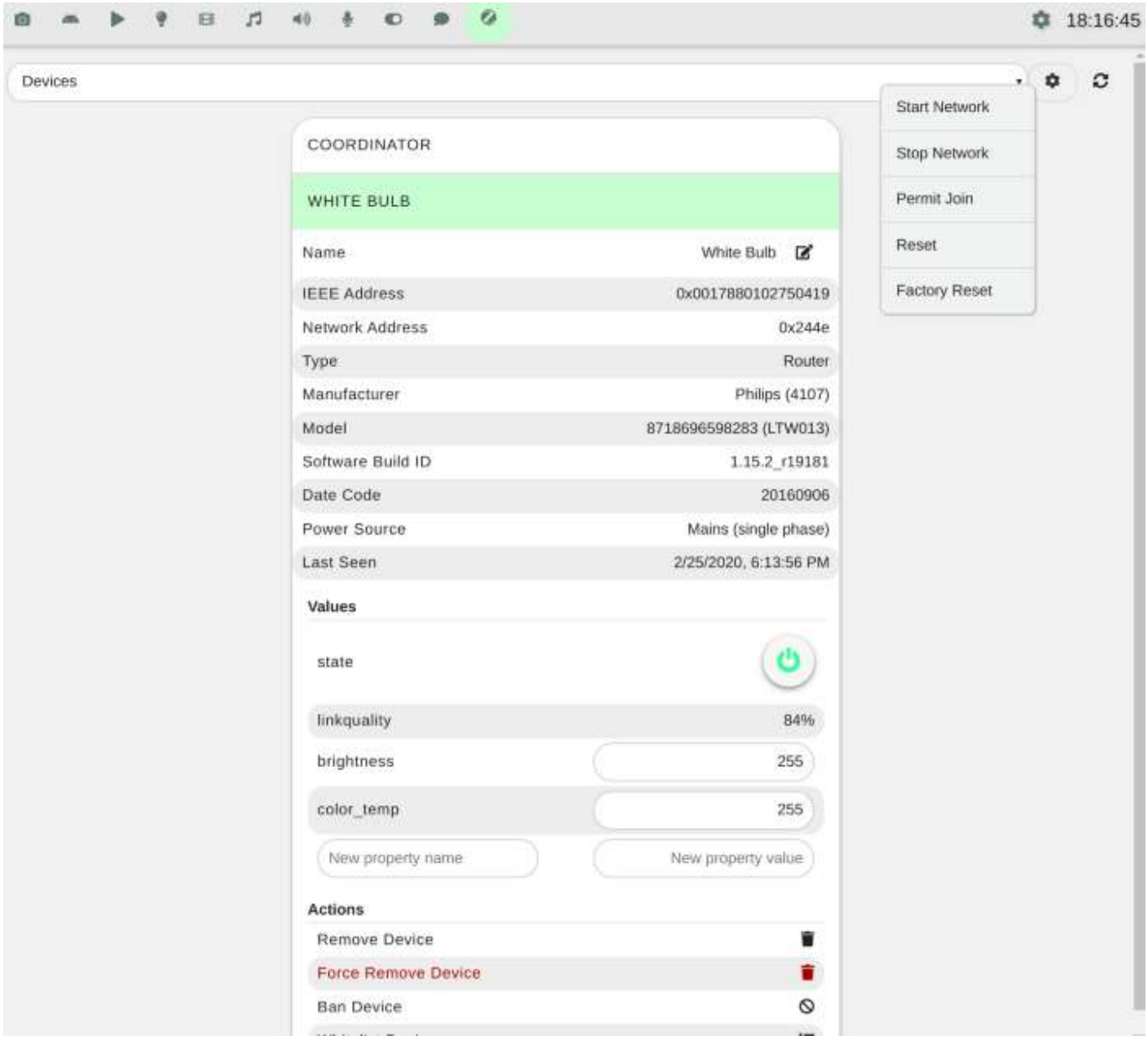
```
[sudo] apt-get install redis-server
[sudo] systemctl start redis.service
[sudo] systemctl enable redis.service
pip install 'platypush[zigbee,http,mqtt]'
```

- Edit your `~/.config/platypush/config.yaml` file to enable the Zigbee and HTTP services:

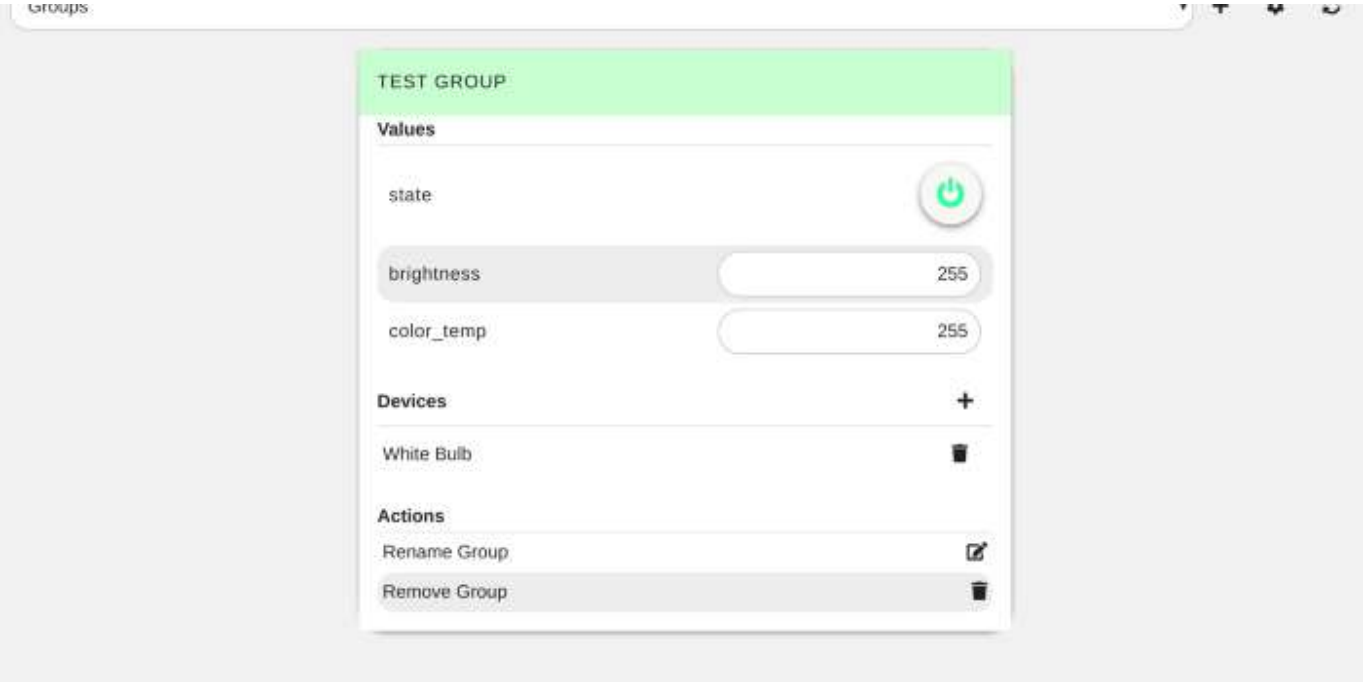
```
backend.http:
    port: 8008

zigbee.mqtt:
    host: localhost
```

```
backend.zigbee.mqtt:
    enabled: true
```

- Start platypush (either by running `platypush` or starting the `platypush.service` systemd service).
- Open `http://host-or-ip:8008/` in a browser. The web panel should include the Zigbee icon in the navigation bar, you'll be able to control your network from there.



platypush Zigbee web panel example with a coordinator and some Philips Hue bulbs attached.



Look mum, no bridges!

You can send requests through the supported API over HTTP, Python code or through whichever platypush backend you have configured:

```
# HTTP request
curl -XPOST -H 'Content-Type: application/json' -d '
{
  "type":"request",
  "action":"zigbee.mqtt.device_set",
  "args": {
    "device":"White Bulb",
    "property":"state",
    "value":"ON"}
}' http://localhost:8008/execute

# Python usage
from platypush.context import get_plugin

get_plugin('zigbee.mqtt').device_set(device='White Bulb',
property='state', value='ON')
```

Or hook any custom logic to the supported events:

```
event.hook.OnLightBulbOn:
    if:
        type:
```

```
platypush.message.event.zigbee.mqtt.ZigbeeMqttDevicePropertySetEvent
        device: White Bulb

    then:
        - if ${properties.get('state') == 'ON'}:
            - action: tts.say
              args:
                  text: "The light went on"
```

Congratulations, you're now ready to use your Zigbee devices and build automation without bridges!

Making your own Z-Wave bridge

Making a DIY Z-Wave bridge is even simpler than making a Zigbee bridge, as you won't need a debugger to flash a custom firmware nor an MQTT service in between.

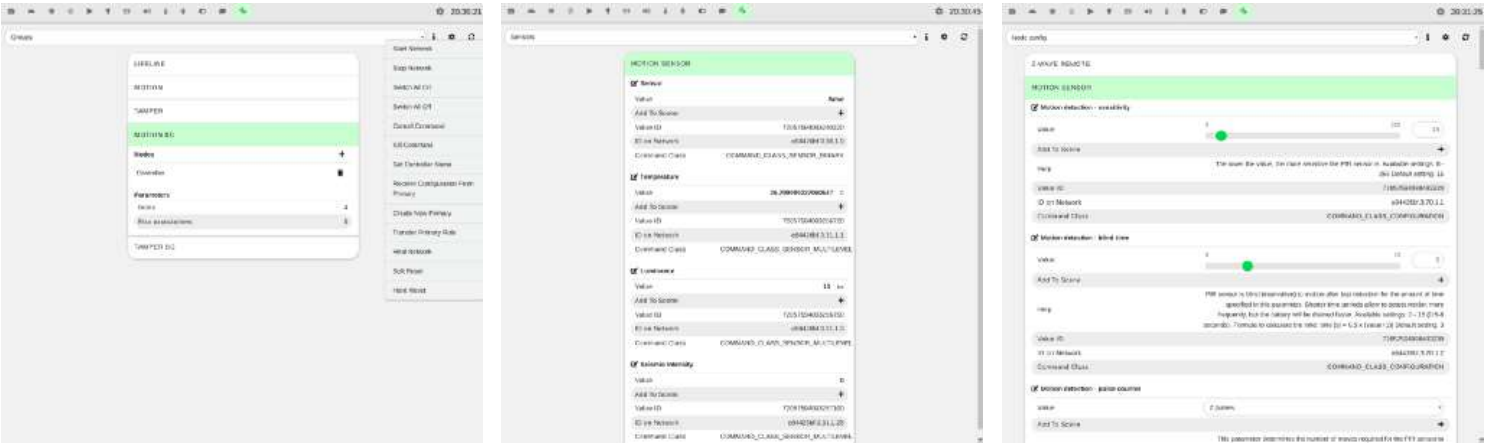
- You'll need a Z-Wave USB adapter dongle. I use this one, but any compatible dongle should work. Take note of where the adapter is mapped on your system — e.g. `/dev/ttyUSB0`.
- Install Redis and platypush with the Z-Wave and HTTP extensions:

```
[sudo] apt-get install redis-server
[sudo] systemctl start redis.service
[sudo] systemctl enable redis.service
pip install 'platypush[zwave,http]'
```

- Configure your integrations in `~/.config/platypush/config.yaml`:

```
backend.http:
    port: 8008

zwave:
    device: /dev/ttyUSB0

backend.zwave:
    enabled: true
```

- Start platypush (by running `platypush` or through systemd service) and point your browser to `http://host-or-ip:8008/`. You'll see a new tab for the Z-Wave integration.



Some screenshots from the Z-Wave web panel.

Each Z-Wave compatible device has its own way of pairing to a network. All you need to do is to press on the `+` button to put the network in pairing mode and then pair your devices within one minute through the procedure referred in the user manual. Since Z-Wave has a stricter protocol and all the compliant devices publish their values using the same format, the Z-Wave interface is much more granular and detailed compared to Zigbee.

You can, of course, send commands to the new network through the available API and subscribe custom hooks on Z-Wave events:

```
# HTTP request
curl -XPOST -H 'Content-Type: application/json' -d '
{
  "type":"request",
  "action":"zwave.get_value",
  "args": {
    "value_label":"Temperature",
    "node_name":"Kitchen Sensor"
  }
}' http://localhost:8008/execute

# Python usage
from platypush.context import get_plugin

get_plugin('zwave').get_value(value_label='Temperature',
node_name='Kitchen Sensor')
```

```
# Example output
{
    "type": "response"
    "target": "http",
    "response": {
        "errors": [],
        "output": {
            "command_class": 49,
            "data": 26.799999237060547,
            "data_items": "Read only",
            "genre": "User",
            "index": 1,
            "is_read_only": true,
            "is_write_only": false,
            "label": "Temperature",
            "node_id": 3,
            "type": "Decimal",
            "units": "C",
            "value_id": 72057594093256722
        }
    },
}

# Trigger a custom action when motion is detected
event.hook.OnZWaveValueEvent:
    if:
        type: platypush.message.event.zwave.ZwaveValueChangedEvent
    then:
        - if ${node['name'] == 'Motion Sensor' and value['label'] ==
'Sensor' and value['data'] is True}:
            - action: tts.say
              args:
                  text: Motion has been detected
```

You should now have all the ingredients to build your custom IoT networks and ditch those bridges for good!

Platypush Zigbee Z Wave IoT Home Automation