Hydration Tracker Project Implementation Guide

Author Ambika Hegde

Overview:

Hydration Tracker is an IoT based project that automatically tracks user's hydration level. Every time a user consumes water or beverages, amount of liquid consumed is recorded. Data is sent to cloud, where the business logic resides on a server-less framework. Received data is processed by the framework and written to database for further reference.

Hydration tracker project involves 3 major components:

1. IoT device equipped with sensors to measure weight of the liquid consumed (hydration tracker device)
2. Business logic that handles device data and user data
3. Database that stores the processed data

Let us look at each of the components in detail.

Setting up hydration tracker device

Step1: Collect all the hardware components required to set up the device

In our project following components are required to set up hydration tracker device:

a. Raspberry Pi 3 kit used as IoT device
b. Force sensor to measure the weight of liquid
c. Analog to digital converter MCP3008, converts analog output from force sensor to digital output
d. Led for alerting the user
e. 330 ohm and 10k resistors for LED and sensor respectively
f. Breadboard and Pi Cobbler breakout box
g. Soldering gun, wires for connection

Connection diagram and explanation can be found in the below link:

https://acaird.github.io/computers/2015/01/07/raspberry-pi-fsr

Step2: Write program to read sensor value

Complete code for reading sensor value can be found in device_code.zip. We used code from https://github.com/acaird/raspi-scale as a starting point (it wasn't a functioning though). To make the code functional we had to clean up lot of redundant code and rewrite some part of it. The original code uses polling method to read values from sensor. However, this method of constant polling is very inefficient and battery powered devices suffer from power drain. Alternatively, we can use interrupt based method to read sensor values. Sensor value output can be given to a pin which is configured as an interrupt pin. Processor is interrupted only when sensor has a new output. We could not implement interrupts due to lack of time (planned for future).

Note: If the HW interfacing is new to you, try to turn LED on off first. It will give some confidence before working of force sensor. Follow the link below

https://projects.raspberrypi.org/en/projects/physical-computing/4

Step3: Set up the connection and read sensor value

After following the instruction provided in the link connected components must look something similar to this.

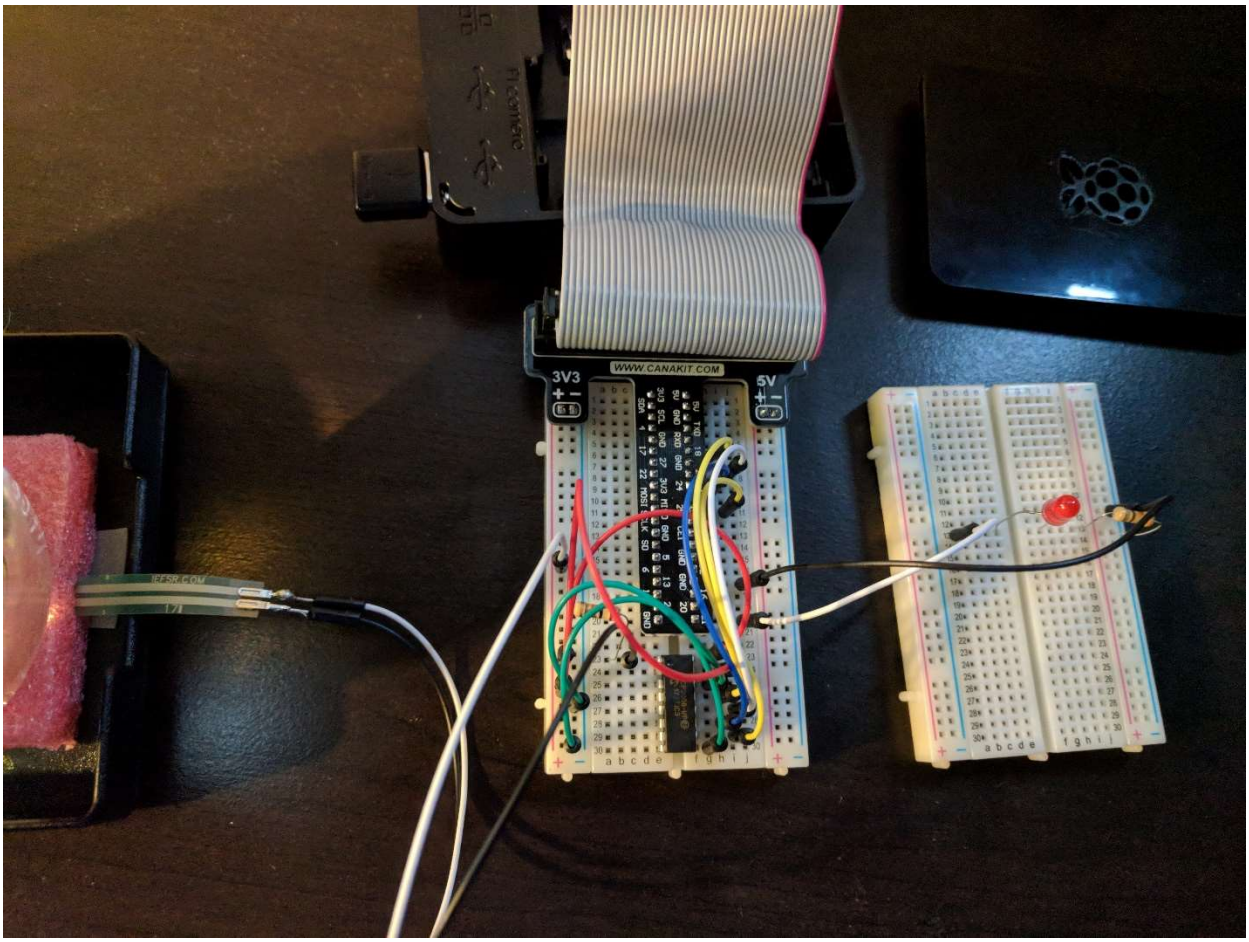


Figure 1: Connection diagram

To read the sensor value from force sensor, run the command ‘python scale-raw.py’. You will be able to view the output something like this

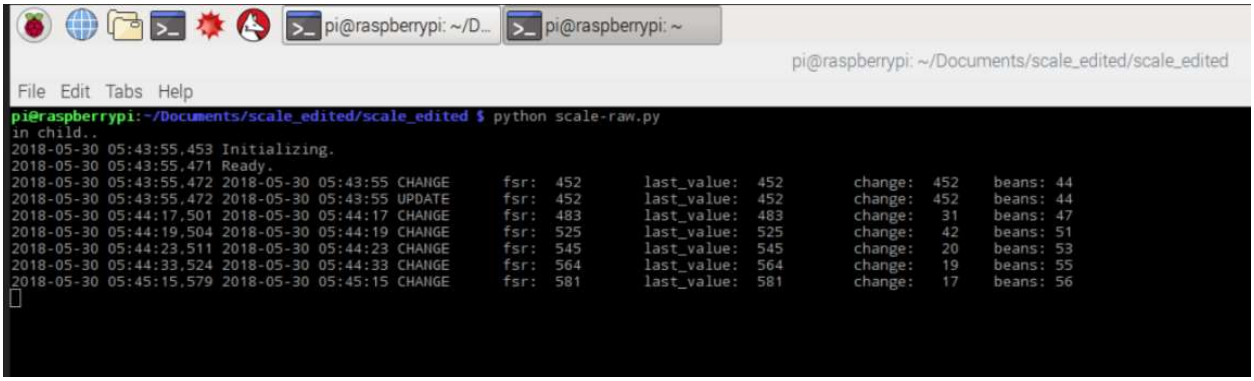


Figure 2: Sensor outputs

Step4: Download AWS Sdk for python and test the connection in AWS IoT console

To connect our IoT device to the cloud we will be using AWS IoT device SDK for Python. Follow the below steps to complete this section

1. Follow the below tutorial to set up your Raspberry Pi as the IoT device
   https://docs.aws.amazon.com/iot/latest/developerguide/iot-sdk-setup.html
2. Download the AWS IoT Device SDK for Python from the following link
   https://github.com/aws/aws-iot-device-sdk-python
3. Test your set up by sending messages from your device to the cloud and receiving messages on your device from cloud. Test the setup using AWS IoT Console

Step5: Merge the AWS IoT device SDK code with sensor module code

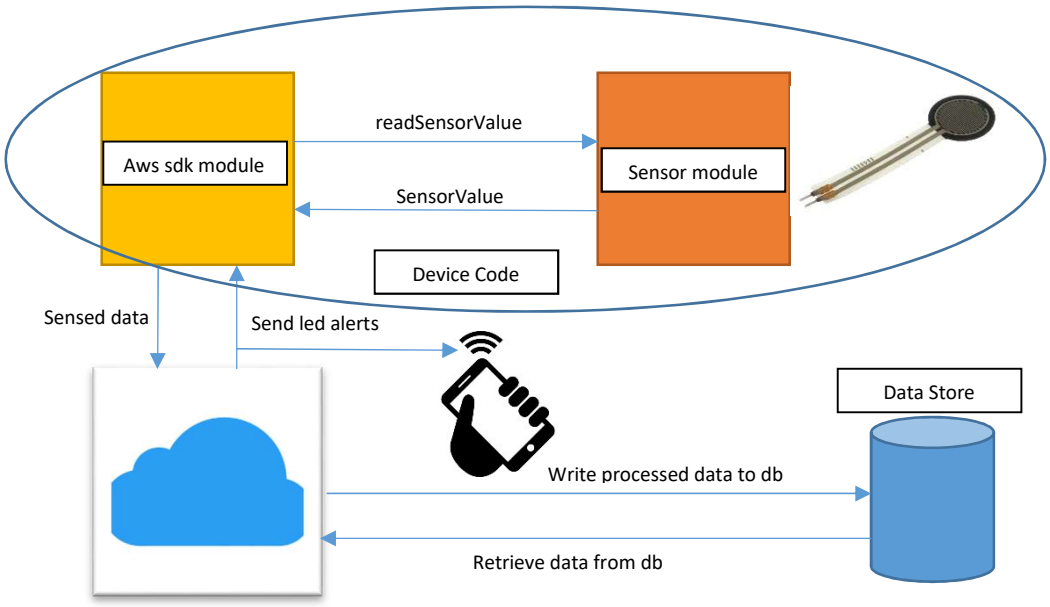


Figure 3: Overview diagram

In this step, we will merge the Sdk code with Sensor module code that we have tested separately in the above sections.

Note: Device code for raspberry pi and sensor module code is present in zipped file Pi_Code.zip. Follow the instruction in README.txt file to execute the code.

Now you will be able to call the sensor module from sdk and read the sensor values. After doing some pre-processing and calibration of the sensed values, AWS sdk module will send the data to cloud by publishing to MQTT topic 'waterConsumed'. Overview diagram given in Figure 4 explains the control and data flow in hydration tracker application.

Messages can be viewed in AWS IoT console by subscribing to 'waterConsumed' topic.

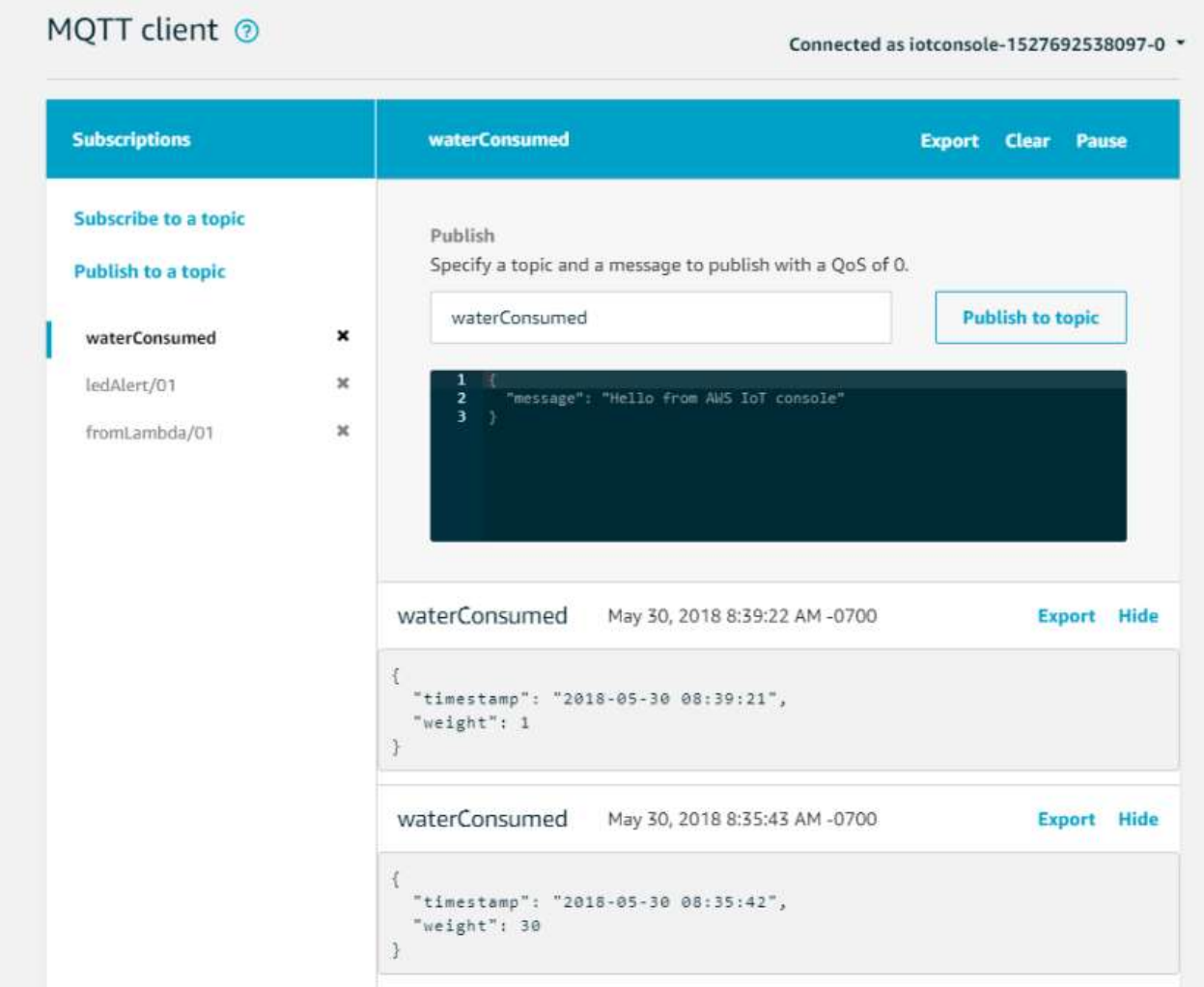


Figure 4: Messages received in 'waterConsumed' topic

Setting up the server

For the hydration tracker project, server side business logic is implemented using AWS lambda functions which are core to serverless computing paradigm. To know more about serverless computing and applications, please refer to the link below.

https://aws.amazon.com/serverless/

We recognized 3 main functionalities of the server in this project.

1. Receiving device data, processing the data and storing it into database
2. Alerting the user if user is not drinking enough water in accordance with their hydration goal
3. Resetting the water consumption data everyday

So we created following 3 lambda functions that handle the above mentioned functionalities

1. Hydrationtracker
2. Notifyuser
3. Resetwater

Note: Each lambda function is associated with a 'role' that can be created in I AM console. Role specifies the policies for lambda. Specifies what services lambda function has access to. For example, hydrationtracker can access AWS RDS, AWS IoT etc. For the project we are using 'accumulatedweight' role. Some of the policies associated with this policy can be found on the right side of the following figure.

Lambda function hydrationtracker

Trigger: AWS IoT

Invokes: AWS RDS, AWS IoT

How is the hydrationtracker function triggered?

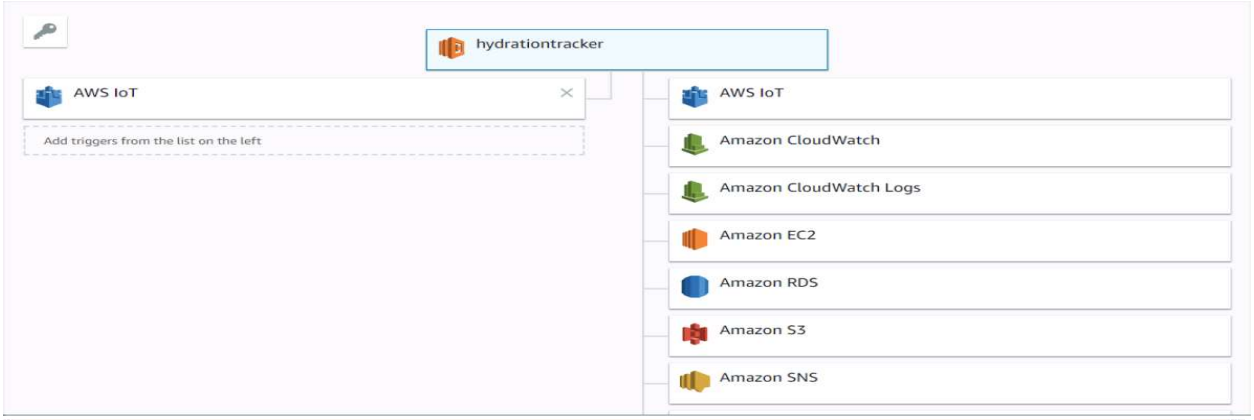
Hydration tracker function is triggered upon the reception of messages to the topic 'waterConsumed' topic. IoT device publishes the message to topics whenever there is new sensor value from the force sensor. Inorder to enable AWS IoT as a trigger, we have to create a rule. For the hydrationtracker function we have created a rule called 'waterRule', details of which can be found in Figure 5 below.

What is the functionality of hydrationtracker?

- Upon trigger, hydration tracker reads the incoming message that is in 'json' format

```
{
  "timestamp": "2018-05-30 08:39:21",
  "weight": 1}
```

- Extracts the weight (amount of water consumed by user)
- Pulls the latest data from water_table corresponding to the user
- Adds the weight and water_table weight, writes the new weight back to database



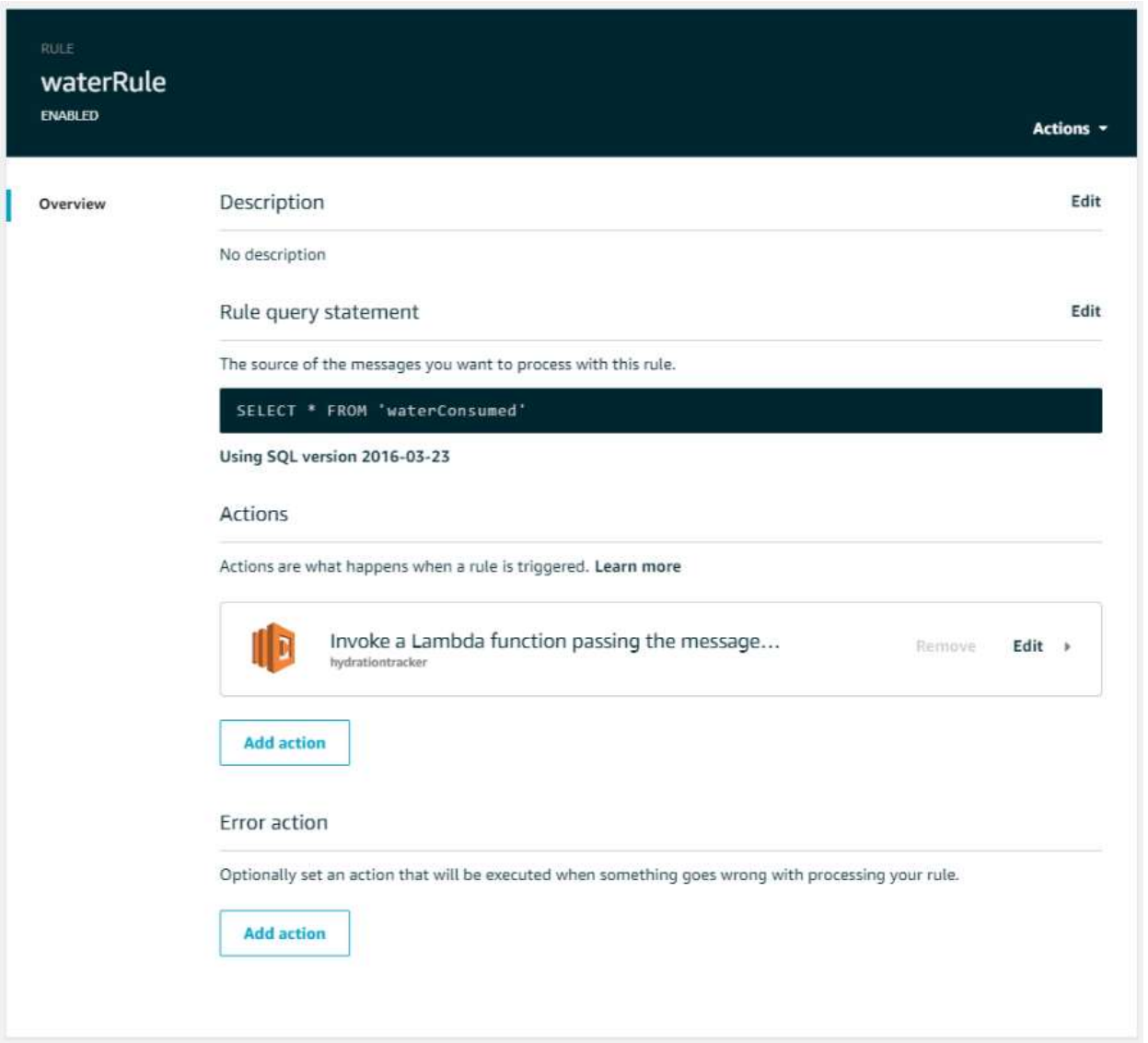


Figure 5: Rule to invoke lambda function hydration tracker upon reception of message in 'waterConsumed' topic

Lambda function resetwaterconsumed

Trigger: Cloudwatch Events (Every day 12:00 AM)

Invokes: AWS RDS

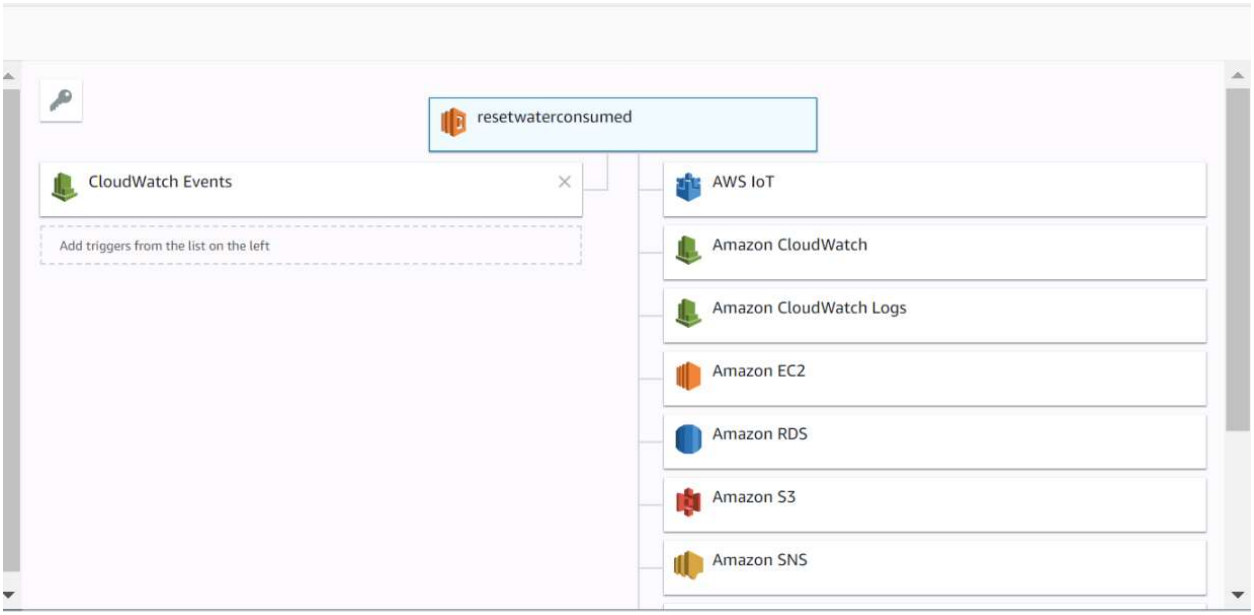
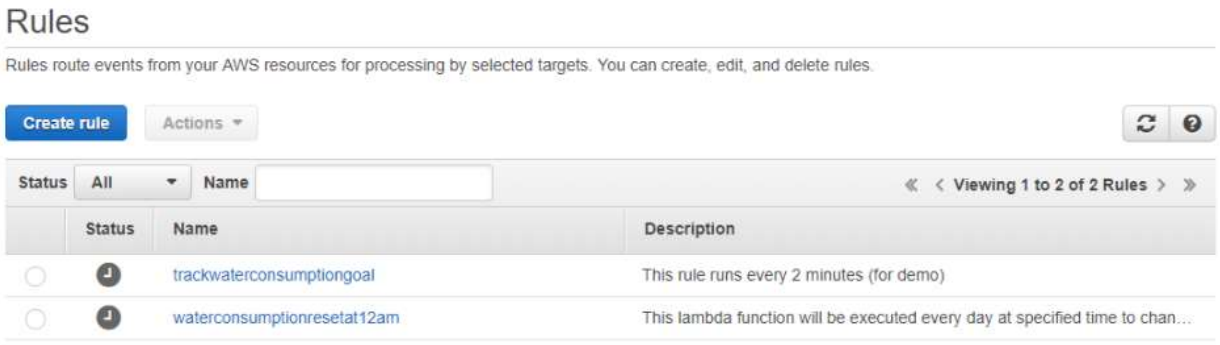


Figure 6 reset water consumed lambda

When the function resetwaterconsumed is triggered via scheduled event trigger, the function resets the water consumed to 0.

Note: inorder to set cloudwatch events, we have to set up Rules in Cloudwatch Management console. For each rule, specify the schedule using either a *cron* or *rate* expression. Specify the function you want to invoke using this rule. For example, *trackwaterconsumptiongoal* invokes notifyuser lambda function. It is important to enable the rule before testing your application.



Lambda function notifyuser

Trigger: Cloudwatch event scheduled at 2 minutes (for demo), in real world may be once every 30 min

Invoke: AWS IoT and AWS SNS

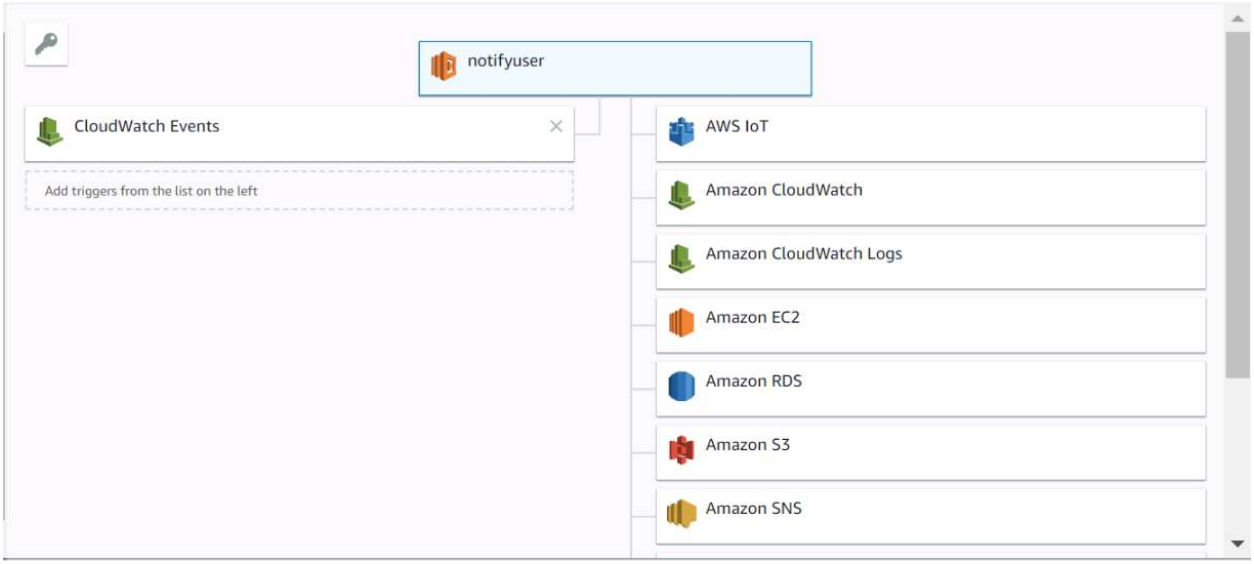


Figure 7 Notify user lambda

When the notifyuser function is invoked it performs following actions

1. Read the last water consumed event from water_table
2. Check if the time elapsed since last event is > T (3 minutes for the purpose of demo)
3. If time_elapsed > 3 minutes, send Led alert to the device by publishing to ledAlert/<userid> topic

Figure 7 shows the behavior upon invoking of notifyuser function

Notice that in the following Figure 8 LED alerts start appearing only when there are no events for 3 or more minutes. Last event observed in waterConsumed topic is at 8:39AM. So alerts are issued when the

lambda function does not observe events for more than 3 minutes.

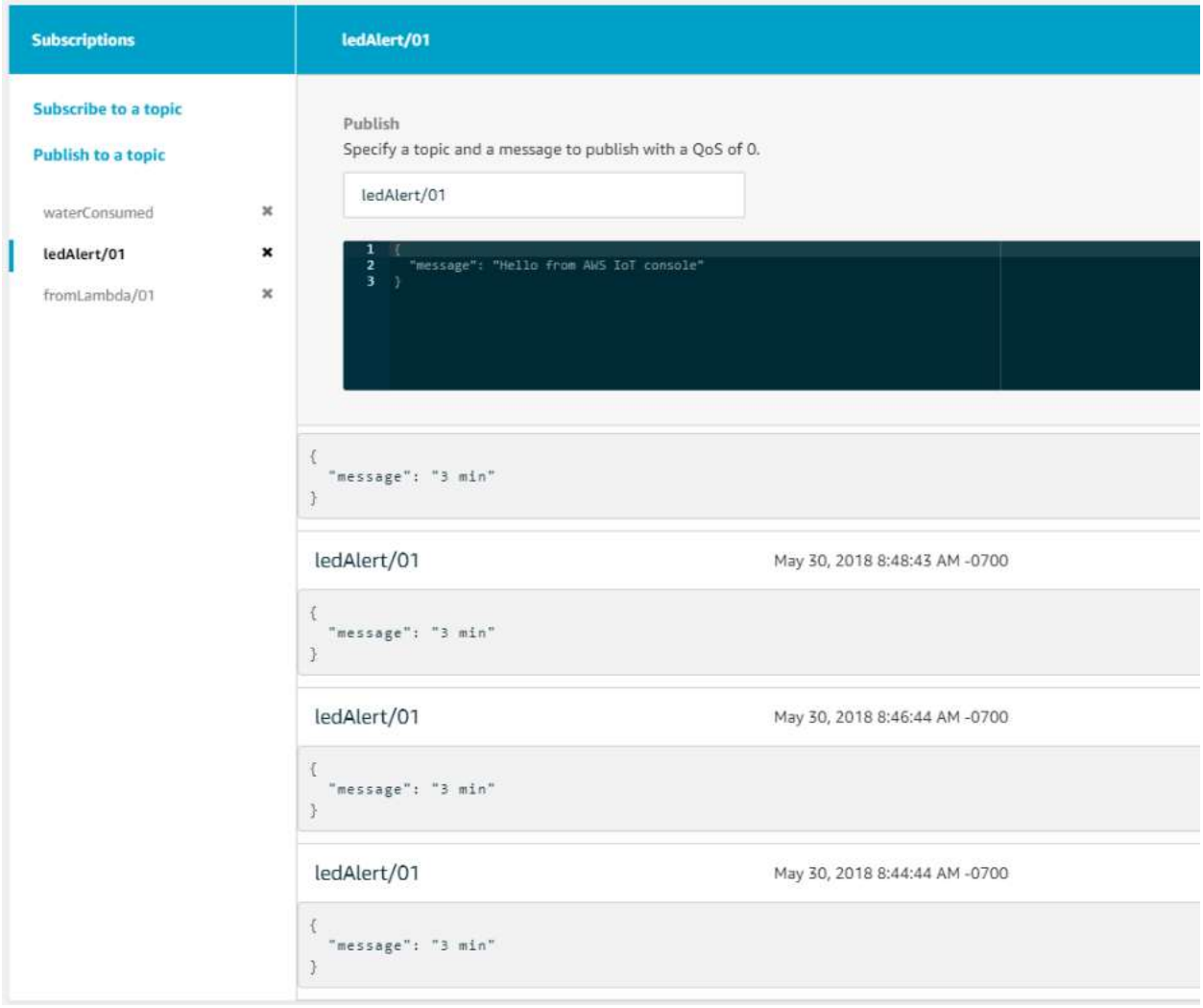


Figure 8 LED alerts

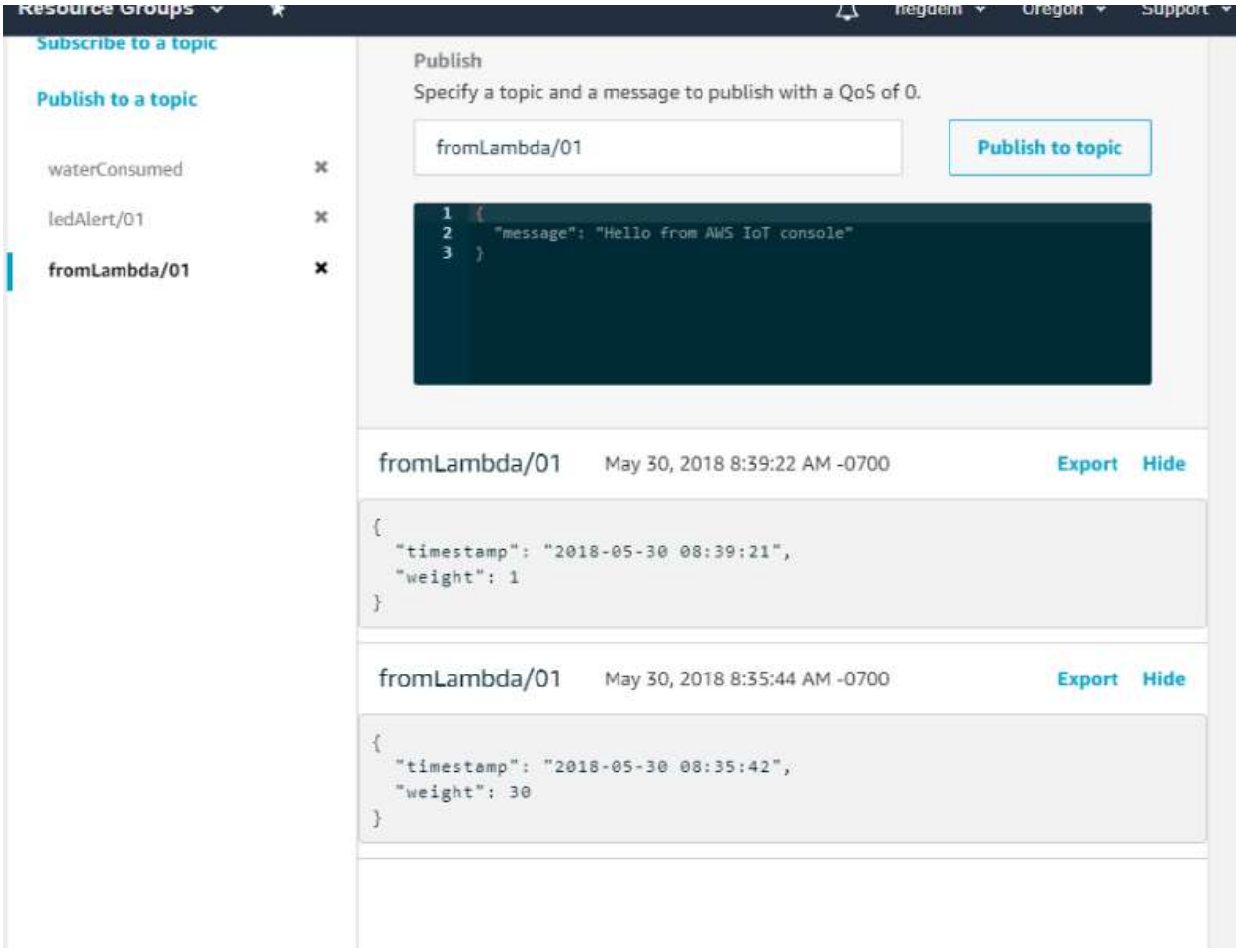


Figure 9 Testing hydrationtracker function

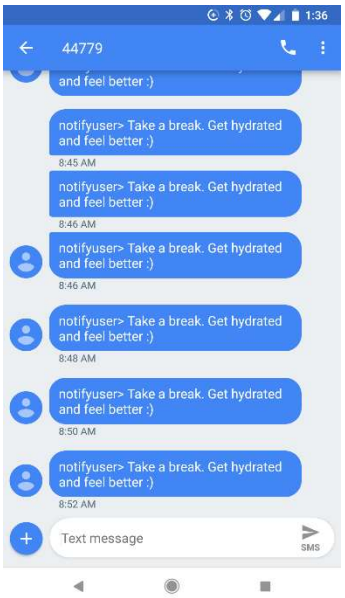
Setting up database and tables

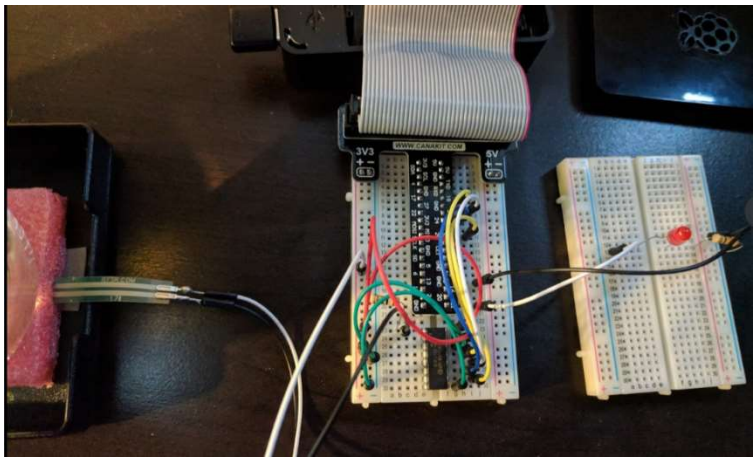
We use AWS RDS Mysql database to store and manage data. AWS provides db2-micro instance for free tier accounts. For the development of the project free tier services are more than enough.

Visualization of data

We are using grafana to visualize data stored MySQL database. Grafana has an option to show the time series data in a neat graph. Daily Weekly and monthly data can be viewed.

Alerts and notifications





At the moment, there are two types of alerts/notifications are sent to the user.

1. Led alert: led alert is sent by publishing to ledAlert/<userid> topic from notifyuser lambda function. IoT device subscribes to the ledAlert/<userid> topic and whenever there is a message in this topic, led is switched on off for 10s. This is an unobtrusive way of notifying user to get hydrated. Basically user need not check mobile phones or wearable inorder to consume water.
2. Text SMS is sent to the user using AWS SNS topics