Title: MakeSense1

Country: Singapore
University/Institute:
Singapore Polytechnic

Members: Pan Ziyue

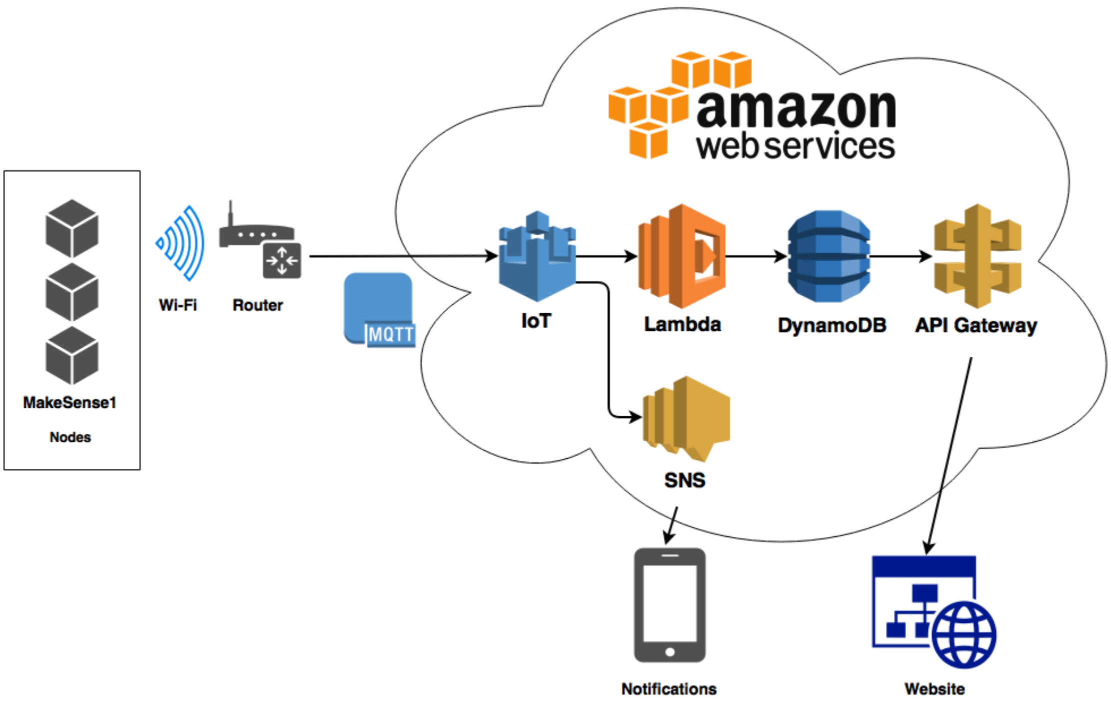
Wei Wen Jie

Mentor: Teo Shin Jen

Foreword and Overview

MakeSense1 is significantly different from traditional applications or devices as it is a complete system, spanning from individual hardware nodes to Amazon's Web Services (AWS) cloud computing platform for data gathering, computation and display. This project utilizes a large variety of code libraries and requires installation prior to compilation. Due to the complexity of this project, a download link is provided below for you to download all of the source code, code libraries, as well as STL object files for 3D printing of the enclosure.

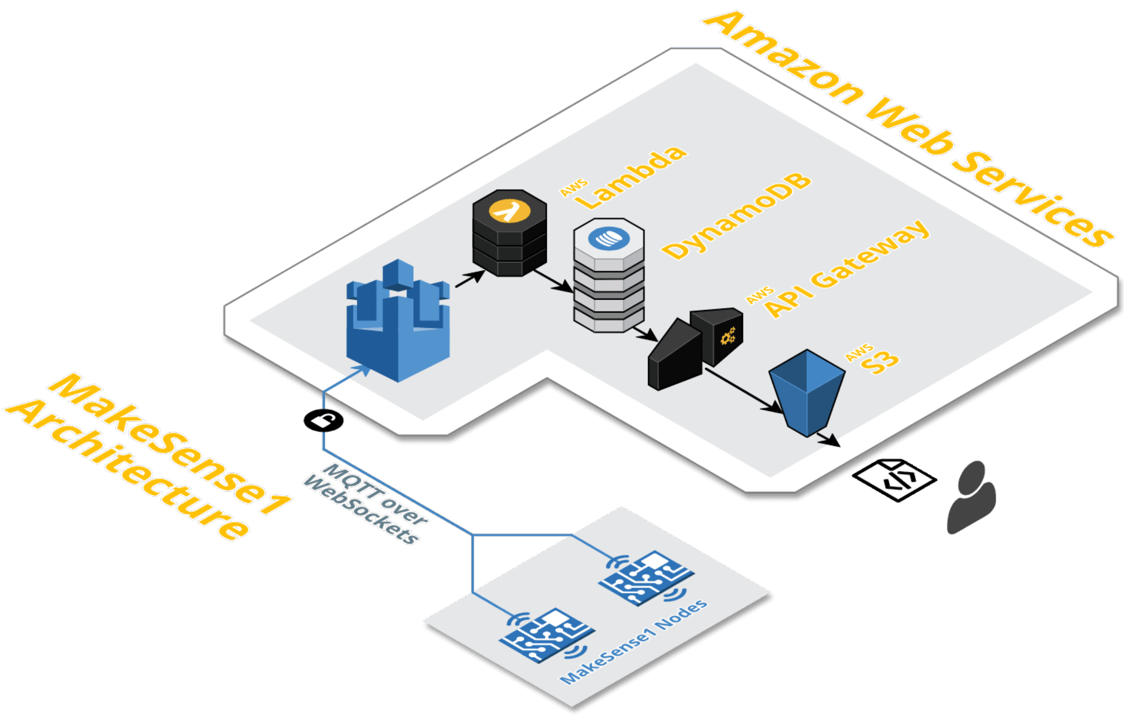
Download link: https://github.com/SPWwj/AWS-IOT-Landslide-Proj



The MakeSense1 includes 3 subsystems: Sensing, Cloud Computing and Visualization.

The Sensing Subsystem uses the SPEEEduino, a custom designed microcontroller board, and interfaces with 3 sensors, namely: rain, IMU and soil moisture sensors, to sense environmental parameters, and transmits acquired data to AWS IoT over the MQTT protocol.

The user plants MakeSense1 nodes at the location to be monitored. Once turned on, the nodes will connect to a communication network as configured. In our case, we are using Wi-Fi, but it can also be changed to SigFox or LoRa as our design is modular and reprogrammable.



The Cloud Computing Subsystem uses Amazon's Web Services' various services, such as AWS IoT for data capture, DynamoDB for data storage and Lambda for data processing. AWS IoT receives data from the MakeSense1 nodes that triggers a Lambda function which writes the data into DynamoDB for storage and computes a new risk factor for that particular node.

Risk factor computation is one of the key objectives of this project. The risk factor R is computed with roll and pitch of the entire node as well as rainfall and soil moisture.

$$R = \frac{\frac{roll + pitch}{4500} + \frac{(|rain - 1023|) + (|soil - 1023|)}{1446 * 50}}{100}$$

<u>**Disclaimer: We open sourced MakeSense1 to enable environmental scientists, researchers and users to evaluate their algorithms and findings on our device. The risk factor computation algorithm shown above is not a scientifically validated algorithm, and requires formal scientific inquiry to improve the accuracy of the algorithm.**</u>

The Visualization Subsystem is written in pure HTML/CSS/JavaScript, and can be hosted on any static webpage host, such as Amazon S3, and relies on Google Maps for providing mapping data and D3.js for rendering individual nodes on the map. It accesses the data from DynamoDB using Amazon's API Gateway.

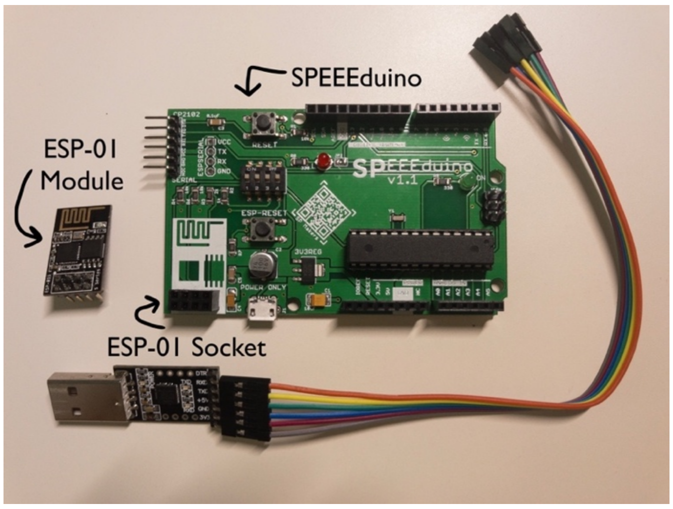
Setting Up

This section explains how to setup a complete MakeSense1 system. If you're not intending to build the hardware system, skip to the Amazon Web Services section.

SPEEEduino (if you do not own a SPEEEduino you can use an Arduino Uno)

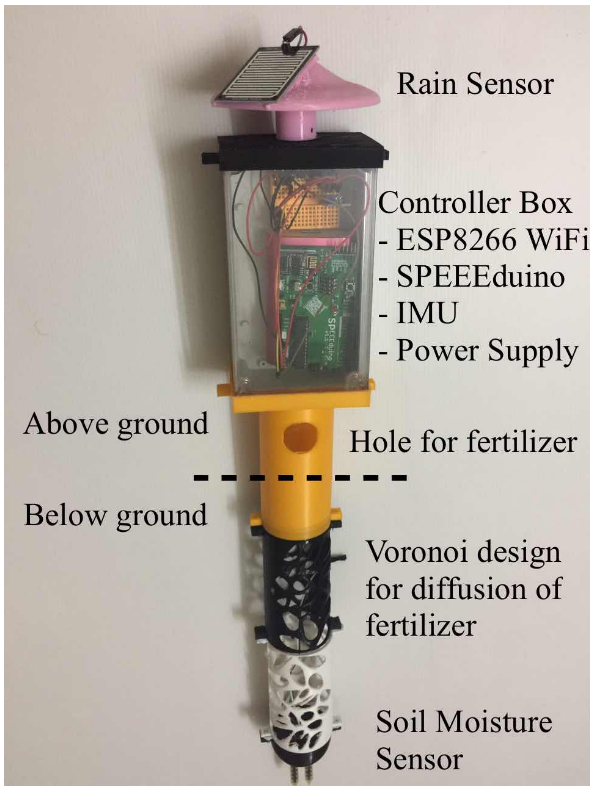
You will need an IP65 waterproof enclosure, a SPEEEduino unit, an ESP-01 ESP8266 Wi-Fi module, a GY-85 IMU module, a soil moisture sensor, and a rain sensor for this part. You may also need connecting wires and a mini-breadboard.

Install the necessary Arduino libraries provided in Arduino Source Code folder in the zip file. Upload ***ESP8266_Side.ino*** to the ESP8266 Wi-Fi module and ***Arduino_Side.ino*** to the SPEEEduino using the Arduino IDE, making sure to enter your Wi-Fi router name and password, your AWS IoT credentials and the desired device ID into the source code (refer to page 5 and 11 of the Source Code List). Connect the ESP8266 Wi-Fi module to the SPEEEduino's ESP-01 socket.



Connect the GY-85 IMU, soil moisture sensor, and rain sensor to the SPEEEduino according to the pin mapping in ***Arduino_Side.ino*** file. You may need jumper wires and a mini-breadboard to connect these components together.

Mount the SPEEEduino, GY-85 and a portable power bank into the IP65 waterproof enclosure. 3D prints the enclosure from the STL files provided in the '3D Printing' folder of the zip file and assemble the components as shown in the diagram below.



Amazon Web Services

You will need an Amazon Web Services account to setup this part. You will need to set up AWS IoT, create a Thing called 'MakeSense1' and note down the details, namely the REST API endpoint URL, MQTT update topic, IAM key, IAM secret, AWS region and the MQTT topic.

Create 2 DynamoDB tables, one called 'devices' and the other called 'entries', with the primary key of each table being 'deviceID' and 'entryUUID' respectively.

The 'devices' table contains data about the individual nodes. You have to fill in the devices table manually before deploying for the system to work. The table below is a sample of what the 'devices' table should look like when it is populated with data. ***deviceID*** is the primary key and each node has a unique ID assigned to it. ***lastUpdated*** contains the last updated UNIX timestamp of the node. ***lat*** and ***long*** indicate the latitude and longitude of the node. ***riskFactor*** is the risk of landslide according to sensor data, computed by a AWS Lambda script.

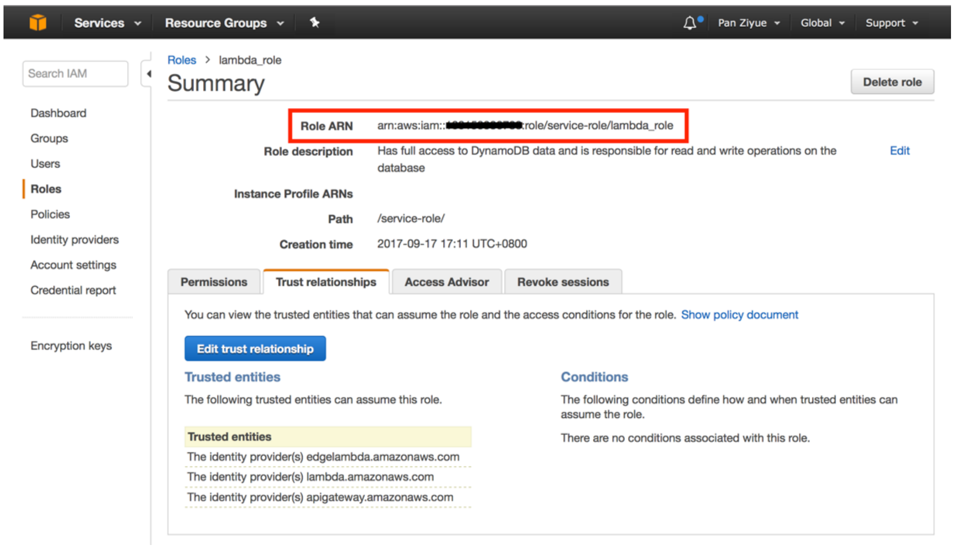
deviceID	lastUpdated	lat	long	riskFactor
1	1505404224	1.353197	103.777876	0.3
0	1506002919	1.350065	103.778273	0.019

The ‘entries’ table contains data of every single sensor node at any given instance. A simplified version of the ‘entries’ table is shown below, the actual table has more raw sensor data. ***entryUUID*** is a uniquely identifying randomly generated UUID for every single entry, and is the primary key of the table. ***deviceID*** is the ID of the device for a given data point.

entryUUID	deviceID	roll	pitch	rain	soil
ff2115f8-2644	0	140	320	333	325
b9112e89-0712	1	140	319	343	233

After setting up DynamoDB, you will need to create a new IAM role and assign it to have AmazonDynamoDBFullAccess and CloudWatchLogsFullAccess permissions so as to allow it to access DynamoDB and API Gateway. The Trust Relationship policy also needs to be updated to the one shown in the Source Code List, Cloud Computing Subsystem section.

Copy down the Role ARN as shown below for setting up the API Gateway later on.



Summary

Role ARN arn:aws:iam::[redacted]:role/service-role/lambda_role

Role description Has full access to DynamoDB data and is responsible for read and write operations on the database

Instance Profile ARNs

Path /service-role/

Creation time 2017-09-17 17:11 UTC+0800

You can view the trusted entities that can assume the role and the access conditions for the role.

Trusted entities

The following trusted entities can assume this role.

Trusted entities
The identity provider(s) edgelambda.amazonaws.com
The identity provider(s) lambda.amazonaws.com
The identity provider(s) apigateway.amazonaws.com

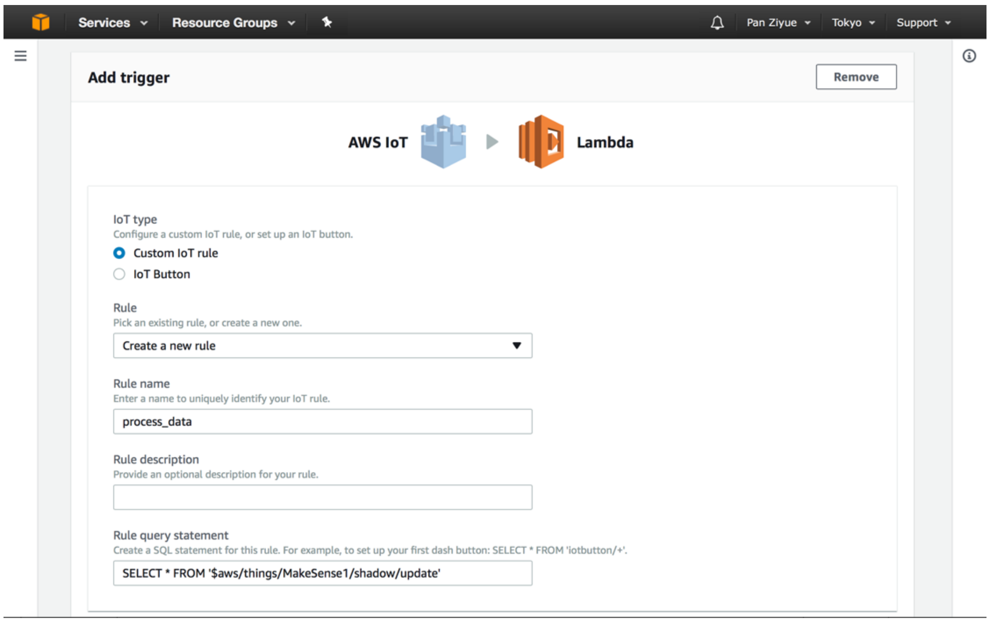
Conditions

The following conditions define how and when trusted entities can assume the role.

There are no conditions associated with this role.

You can now set up AWS Lambda which bridges AWS IoT and DynamoDB. Create a function from scratch using the AWS Lambda console and add AWS IoT as the trigger, creating a new rule for AWS IoT with the Rule Query Statement which forwards all data from every update request to the Lambda function and make sure that Enable Trigger is checked.

```
SELECT * FROM '$aws/things/MakeSense1/shadow/update'
```



Add trigger

IoT type
Configure a custom IoT rule, or set up an IoT button.
● Custom IoT rule
○ IoT Button

Rule
Pick an existing rule, or create a new one.
Create a new rule

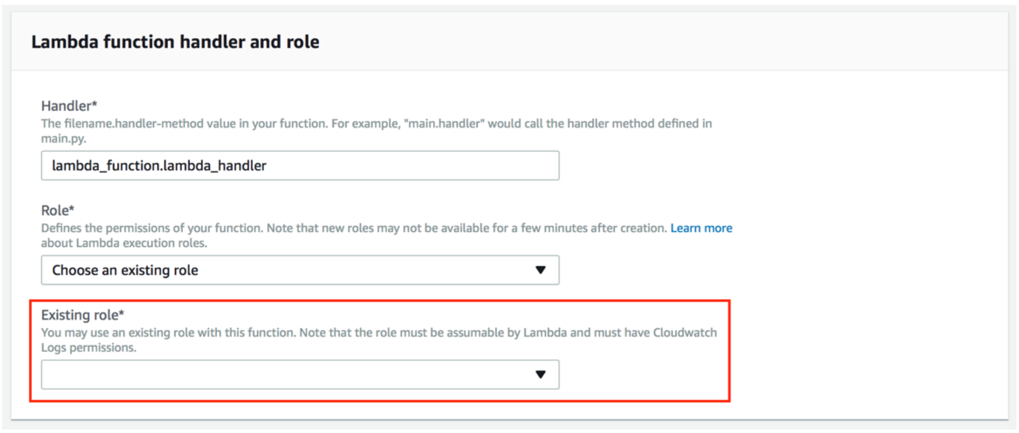
Rule name
Enter a name to uniquely identify your IoT rule.
process_data

Rule description
Provide an optional description for your rule.

Rule query statement
Create a SQL statement for this rule. For example, to set up your first dash button: SELECT * FROM 'iotbutton/+'.
SELECT * FROM '$aws/things/MakeSense1/shadow/update'

Now, configure the function runtime to be Python 3.6, and copy-paste ***write_timestamp_calculate_risk.py*** into the Lambda function code area.

Select the IAM role you previously created in the Existing role field to grant the Lambda function sufficient permissions to access the database.



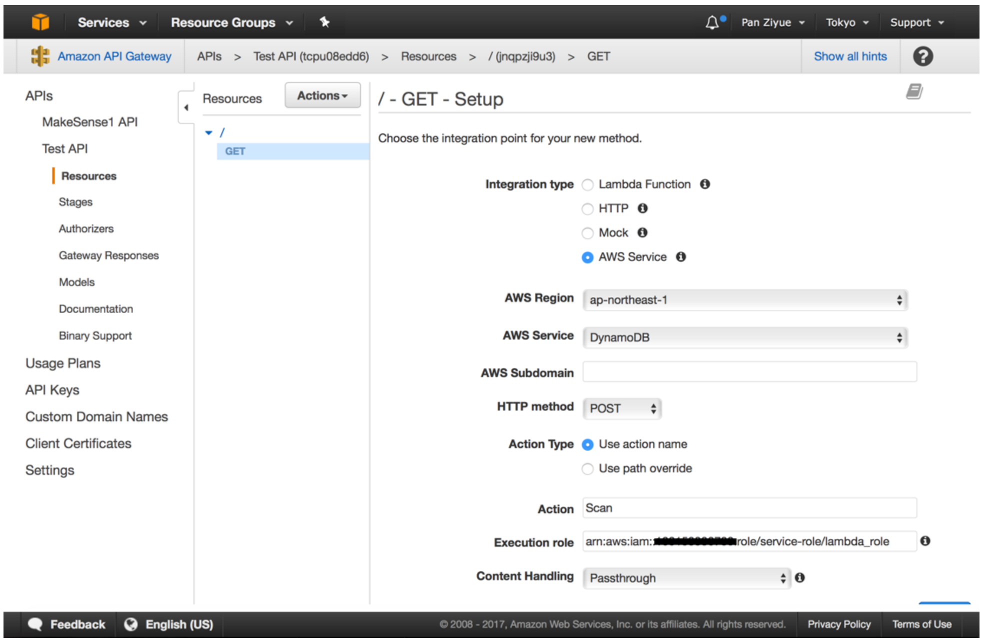
Lambda function handler and role

Handler*
The filename.handler-method value in your function. For example, "main.handler" would call the handler method defined in main.py.
lambda_function.lambda_handler

Role*
Defines the permissions of your function. Note that new roles may not be available for a few minutes after creation. Learn more about Lambda execution roles.
Choose an existing role

Existing role*
You may use an existing role with this function. Note that the role must be assumable by Lambda and must have Cloudwatch Logs permissions.

For the last step, create an API using AWS API Gateway by creating a GET method on the '/' resource, and filling in the form according to the picture below, inserting your Execution Role (ARN) from the step on page 6.



/ - GET - Setup

Choose the integration point for your new method.

Integration type: Lambda Function / HTTP / Mock / [✓ AWS Service]

AWS Region: ap-northeast-1

AWS Service: DynamoDB

AWS Subdomain:

HTTP method: POST

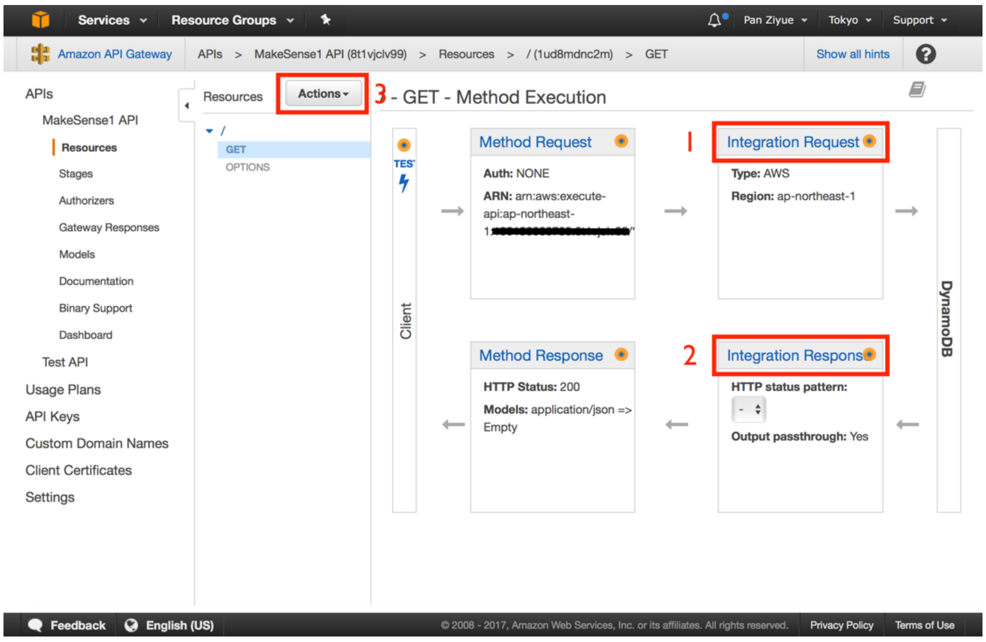
Action Type: [✓ Use action name] / Use path override

Action: Scan

Execution role: arn:aws:iam::[redacted]role/service-role/lambda_role

Content Handling: Passthrough

Configure the Body Mapping Template of the (1) Integration Request and (2) Integration Response using the code provide in "AWS Code" folder. Make sure to configure the Resource to (3) enable CORS (Cross Origin Resource Sharing) so that your browser can download data from the API. Finally, deploy the API.



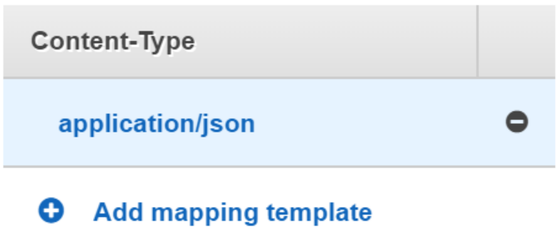
/ - GET - Method Execution

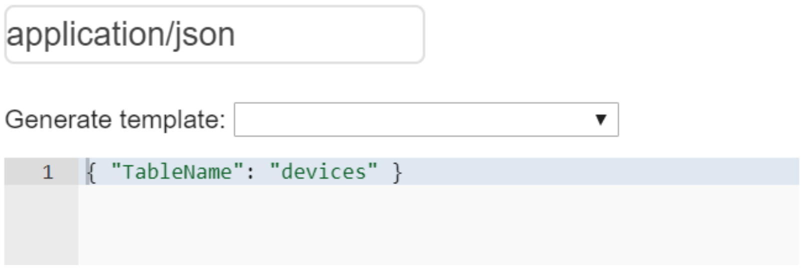
Method Request
Auth: NONE
ARN: arn:aws:execute-api:ap-northeast-1[redacted]

Integration Request
Type: AWS
Region: ap-northeast-1

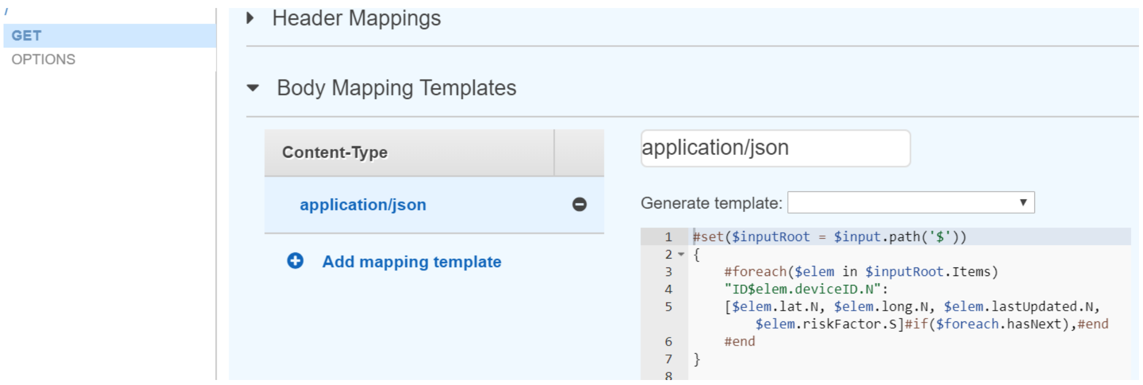
Method Response
HTTP Status: 200
Models: application/json => Empty

Integration Response
HTTP status pattern: -
Output passthrough: Yes



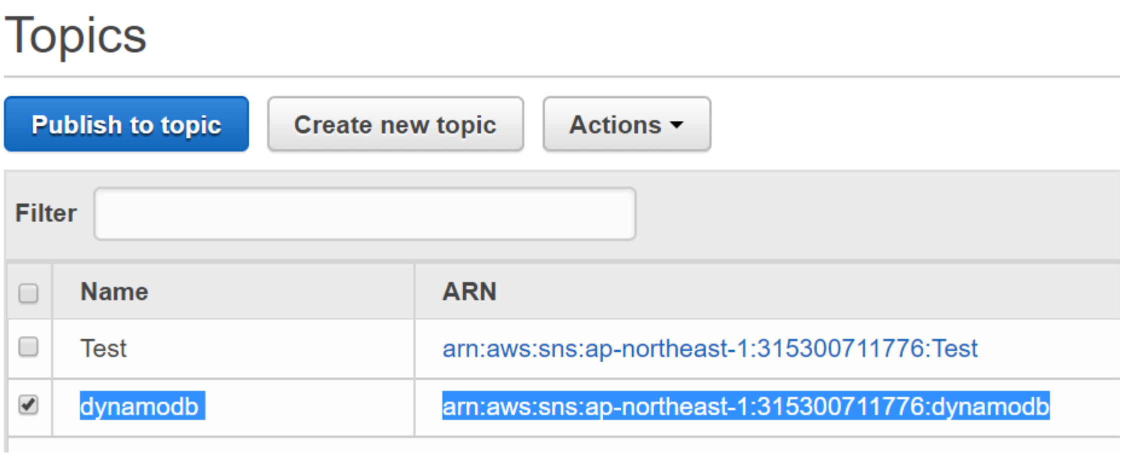


```
{ "TableName": "devices" }
```



```
#set($inputRoot = $input.path('$'))
{
    #foreach($elem in $inputRoot.Items)
    "ID$elem.deviceID.N":
    [$elem.lat.N, $elem.long.N, $elem.lastUpdated.N,
        $elem.riskFactor.S]#if($foreach.hasNext),#end
    #end
}
```

Next go to Simple Notification Service to set up SMS notification
Select Topics and create a new topic called “dynamodb”

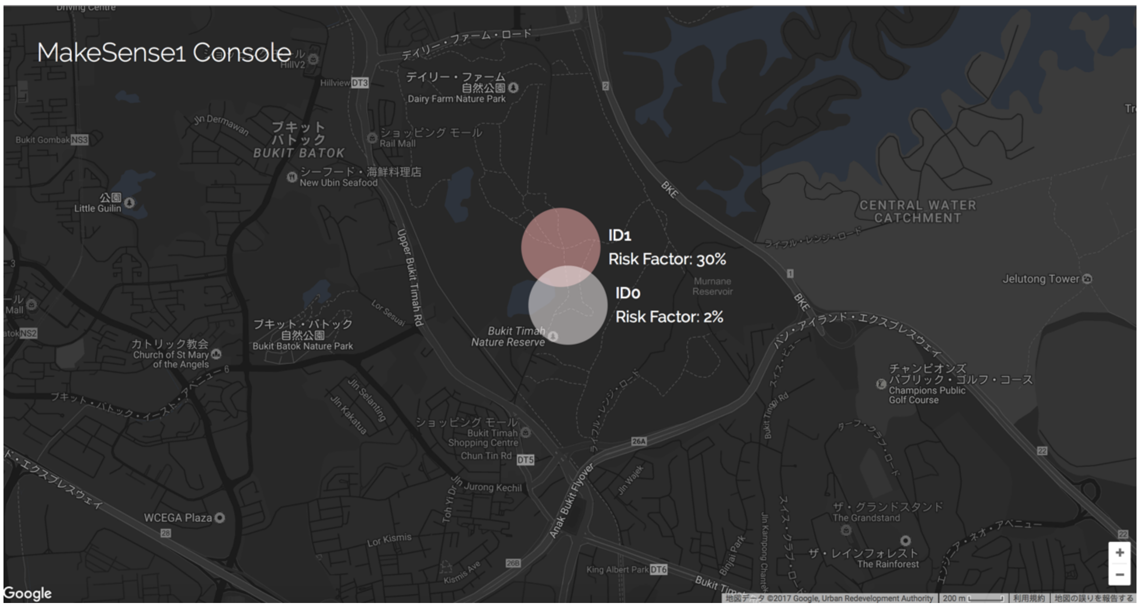


	Name	ARN
☐	Test	arn:aws:sns:ap-northeast-1:315300711776:Test
✓	dynamodb	arn:aws:sns:ap-northeast-1:315300711776:dynamodb

Then tick the checkbox of your topic, under the Actions click Subscribe to Topic, choose SMS and enter your phone number.

Visualization Subsystem

The Visualization subsystem is fairly straightforward to set up, and you don't even have to deploy it onto a server for it to work. Simply tweak the line containing the API URL (looks like `var url = "https://8t1vjclv99.execute-api.ap-northeast-1.amazonaws.com/Production";`) to your API's actual URL and double click ***index.html*** in the 'D3 Visualization' folder of the source code zip file. It should present you with the MakeSense1 Console, which shows all of the nodes and their respective risk factors. In our demonstration unit, we preprogrammed the coordinates of the nodes to be around Bukit Timah Hill, which is the only place in Singapore with significant elevations above sea level. If you wish to deploy it onto a server, Amazon's S3 static hosting service can be used. Now you can power on your device to see data streaming in and protect your property from landslides.

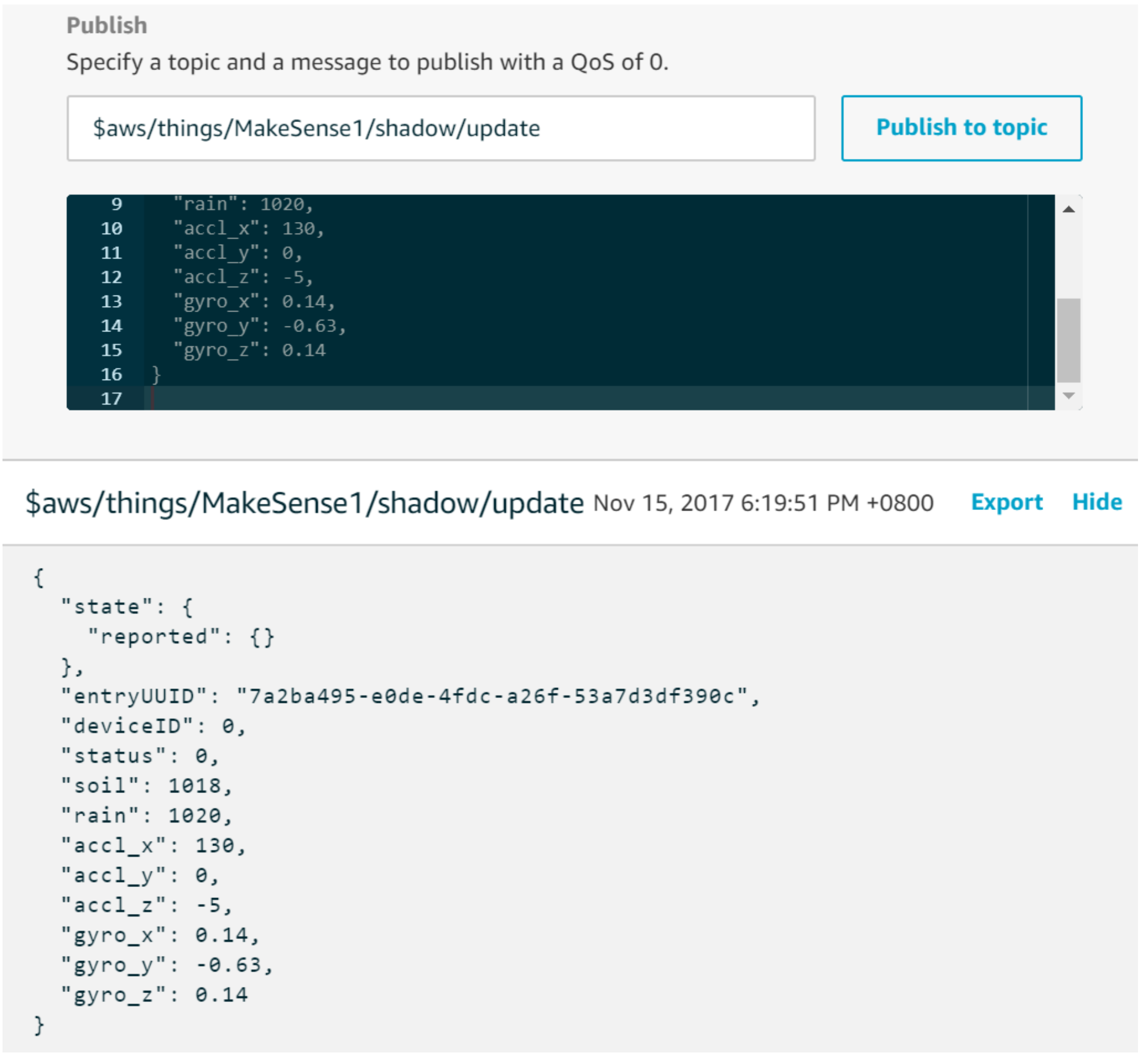


Demo Instructions (Without actual hardware)

If you want to test out the Cloud and Visualization systems without building the actual hardware, follow the instructions below:

1) Fire up the Visualization subsystem by opening the `index.html` file in the D3 folder under our GitHub repository.
2) Open the AWS IoT MQTT client (you must set up your own AWS IoT account, if you are using our AWS Setup, please use our private key in the Zip file).
3) Copy the sample sensor data in D3 folder under GitHub repository to the input field of the MQTT client (and tweak the values to vary the risk factor output).
4) Publish the sensor data to your AWS IoT topic (in our case, it is `$aws/things/MakeSense1/shadow/update`)
5) Enjoy using the MakeSense1!

```
// Sample Sensor Data:
{
  "state": {
    "reported": {}
  },
  "entryUUID": "e55bc185-015b-4864-aea4-648f9f05f0de",
  "deviceID": 0,
  "status": 0,
  "soil": 300,
  "rain": 300,
  "accl_x": 0,
  "accl_y": 0,
  "accl_z": 0,
  "gyro_x": -0.56,
  "gyro_y": -0.28,
  "gyro_z": 0.42
}
```



```
$aws/things/MakeSense1/shadow/update
```

```
  "rain": 1020,
  "accl_x": 130,
  "accl_y": 0,
  "accl_z": -5,
  "gyro_x": 0.14,
  "gyro_y": -0.63,
  "gyro_z": 0.14
}
```

$aws/things/MakeSense1/shadow/update Nov 15, 2017 6:19:51 PM +0800

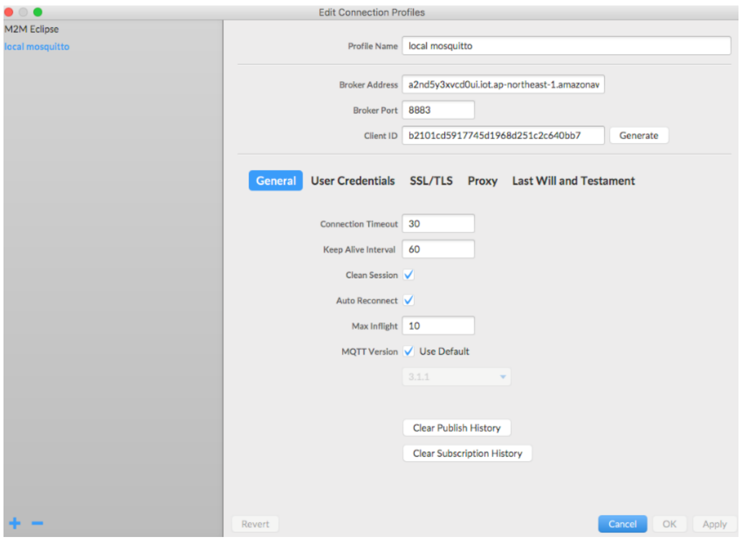
```
{
  "state": {
    "reported": {}
  },
  "entryUUID": "7a2ba495-e0de-4fdc-a26f-53a7d3df390c",
  "deviceID": 0,
  "status": 0,
  "soil": 1018,
  "rain": 1020,
  "accl_x": 130,
  "accl_y": 0,
  "accl_z": -5,
  "gyro_x": 0.14,
  "gyro_y": -0.63,
  "gyro_z": 0.14
}
```

Setting up MQTT client using mqtt.fx

1)

Enter the Broker Address as AWS IOT End point

Broker port 8883


Edit Connection Profiles

M2M Eclipse
local mosquitto

Profile Name: local mosquitto
Broker Address: a2nd5y3xvcd0ui.iot.ap-northeast-1.amazonav
Broker Port: 8883
Client ID: b2101cd5917745d1968d251c2c640bb7

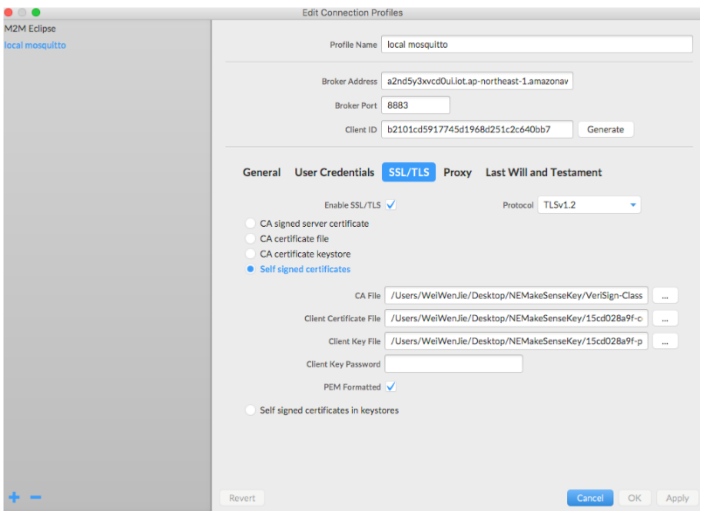
General | User Credentials | SSL/TLS | Proxy | Last Will and Testament

Connection Timeout: 30
Keep Alive Interval: 60
Clean Session: ✓
Auto Reconnect: ✓
Max Inflight: 10
MQTT Version: ✓ Use Default
3.1.1

2)

Go to SSL/TLS tab

Select the Self signed certificates and enter the certs respectively


Edit Connection Profiles

M2M Eclipse
local mosquitto

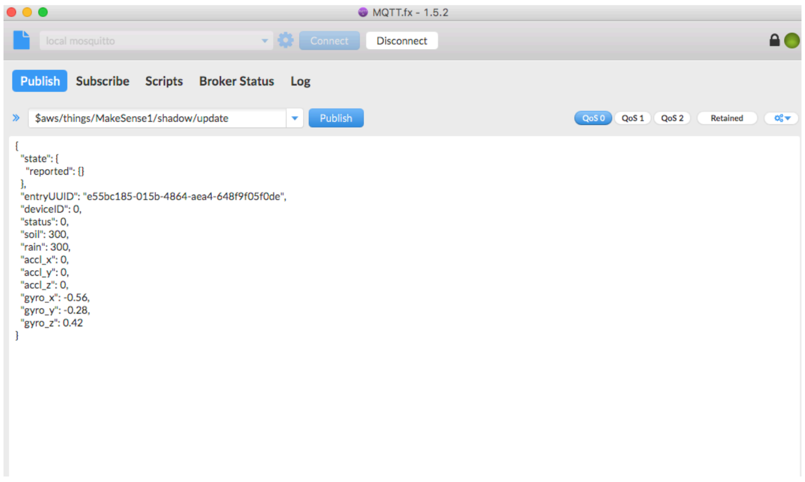
Profile Name: local mosquitto
Broker Address: a2nd5y3xvcd0ui.iot.ap-northeast-1.amazonav
Broker Port: 8883
Client ID: b2101cd5917745d1968d251c2c640bb7

General | User Credentials | SSL/TLS | Proxy | Last Will and Testament

Enable SSL/TLS: ✓
Protocol: TLSv1.2
CA signed server certificate
CA certificate file
CA certificate keystore
● Self signed certificates
CA File: /Users/WeiWenJie/Desktop/NEMakeSenseKey/VeriSign-Class
Client Certificate File: /Users/WeiWenJie/Desktop/NEMakeSenseKey/15cd028a9f-c
Client Key File: /Users/WeiWenJie/Desktop/NEMakeSenseKey/15cd028a9f-p
Client Key Password:
PEM Formatted: ✓
Self signed certificates in keystores

3)

Lastly Publish the sensor data to the Topic “`$aws/things/MakeSense1/shadow/update`”



$aws/things/MakeSense1/shadow/update

```
{
 "state": {
  "reported": {}
 },
 "entryUUID": "e55bc185-015b-4864-aea4-648f9f05f0de",
 "deviceID": 0,
 "status": 0,
 "soil": 300,
 "rain": 300,
 "accl_x": 0,
 "accl_y": 0,
 "accl_z": 0,
 "gyro_x": -0.56,
 "gyro_y": -0.28,
 "gyro_z": 0.42
}
```