

Selected Topics in Industrial Control & Instrumentation (EE5111)

Implement a Simple IoT Pipeline with AWS Cloud Platform and Visualise the Data

Rogatiya Mohmad Aspak Arif (A0179741U)
Department of ECE, National University of Singapore
e0269760@u.nus.edu
+65 84073081

[Medium](#)
[GitHub](#)

Introduction

In a real-time embedded system, the architecture is subdivided to different layers and each of the layer can be replaced without affecting the functionality of overall IoT system. Sensor is sensing and gathering information about environment. Low power embedded processor process data and interface with sensor and wireless transceiver that transfer data to internet gateway. Internet gateway connects to internet and allows data transfer to server. The application is running on server to deliver application specific service to user.

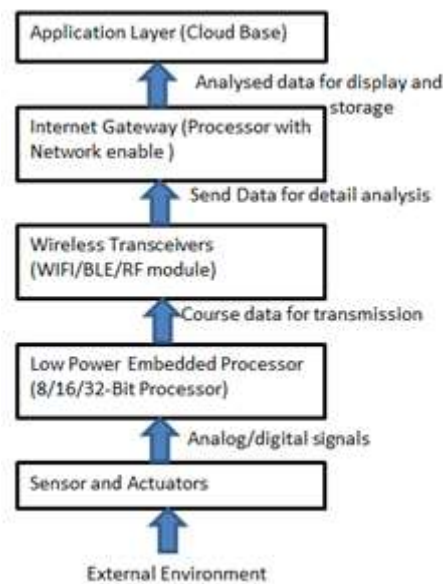


Figure 1: 5 Layers of a simple IOT Architecture

This project is to implement a simple Internet of Things (IoT) pipeline with AWS Cloud platform and visualise the data. AWS IoT provides secure, bi-directional communication between Internet-connected devices such as sensors, actuators, embedded microcontrollers and the AWS Cloud. This enables user to collect telemetry data from multiple devices, and store and analyse the data. We will simulate two small IoT setups through Python that record and push data from two jet engines as shown in Figure 2 to AWS DynamoDB. A step by step guideline is written on how to set up the entire pipeline process.

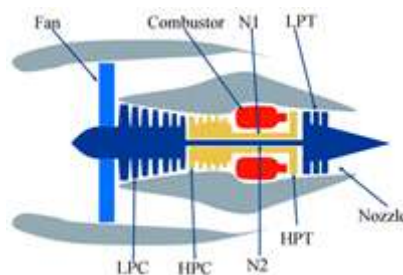


Figure 2: Figure 2 — Jet Engine

In this task, we obtained four jet engines data namely FD001, FD002, FD003 and FD004 in text file format from given, each text file contains a jet engine data operating settings and sensors from Commercial Modular Aero-Propulsion System. The requirement is to publish 2 pre-defined FD001 and

FD002 jet engine data as shown in Figure 3 as IoT Things to AWS. In this arrangement of data, there are a sum of 26 arrangements of data for every line, which is given in the following configurations: id, cycle, os1, os2, os3, sensor1,..., sensor21 which relate to the data collected on the jet engine at each time instance.

FileEditSearchViewEncodingLanguageSettingsMacroRunPluginsWindow?

train_FD001.txt

111-0.0007-0.0004100.0518.67641.821589.70140

2120.0019-0.0003100.0518.67642.151591.821403

313-0.00430.0003100.0518.67642.351587.991404

4140.00070.0000100.0518.67642.351582.791401

515-0.0019-0.0002100.0518.67642.371582.85140

616-0.0043-0.0001100.0518.67642.101584.47139

7170.00100.0001100.0518.67642.481592.321397

818-0.00340.0003100.0518.67642.561582.961400

9190.00080.0001100.0518.67642.121590.981394

10110-0.00330.0001100.0518.67641.711591.24140

111110.0018-0.0003100.0518.67642.281581.75140

121120.00160.0002100.0518.67642.061583.411400

13113-0.00190.0004100.0518.67643.071582.19140

141140.0009-0.0000100.0518.67642.351592.95139

15115-0.0018-0.0003100.0518.67642.431583.8214

train_FD002.txt

11134.99830.8400100.0449.44555.321358.61

21241.99820.8408100.0445.00549.901353.22

31324.99880.821860.0462.54537.311256.76

41442.00770.8416100.0445.00549.511354.03

51525.00050.820360.0462.54537.071257.71

61625.00450.820560.0462.54537.021266.38

71742.00430.8409100.0445.00549.741347.45

81820.00200.7002100.0491.19607.441481.69

91941.99850.8407100.0445.00549.331348.23

1011042.00110.8400100.0445.00549.331356.40

1111142.00290.8400100.0445.00549.811352.72

121120.00150.0010100.0518.67642.701585.52

1311320.00030.7000100.0491.19607.671488.74

1411442.00200.8407100.0445.00549.471352.65

1511510.00380.2513100.0484.05604.571501.77

Normal text file

length:9,082,480lines:53,760Ln:1Col:1Sel:0|0

Unix (LF)UTF-8INS

Figure 3:FD001 and FD002 Jet Engines Data

Introduction to Message Queuing Telemetry Transport (MQTT)

In this assignment, we will be using MQTT as a Client Server to publish/subscribe IoT Things data, MQTT is a messaging transport protocol that is lightweight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in constrained environments such as for communication in Machine to Machine (M2M) and IoT contexts where a small code footprint is required and/or network bandwidth is at a premium. Figure 4 shows the full AWS IoT communication diagram for this assignment to publish data from two jet engines to client server, and AWS DynamoDB would subscribe from it.



Figure 4 : AWS IoT Communication Diagram

Setting Up AWS IoT and Sending Data with Development Computer: The setting process requires registration of an AWS account, an IAM administrator user in the AWS account and a desktop or laptop development computer to work with the AWS IoT console from a web browser, and to push jet engines data into AWS IoT. This computer can be running a Windows, macOS, Linux, or Unix operating system.

Create Thing1 for FD001

Step 1: Create the AWS IoT Policy for FD001 Thing 1

In this step, to allow desktop/laptop as a substitute simulator, to perform AWS IoT operations by creating an AWS IoT policy. X.509 certificates are used to authenticate devices with AWS IoT. AWS IoT policies are used to authorize devices to perform AWS IoT operations, such as subscribing or publishing to MQTT topics. Under IoT Core console, select 'Secure' then 'Policies' to 'Create' a policy for FD001 Thing 1 as shown in Figure 5. Follow the red boxes guide and Enter a 'Name', 'iot:*', '*' and check 'Allow' checkbox respectively.

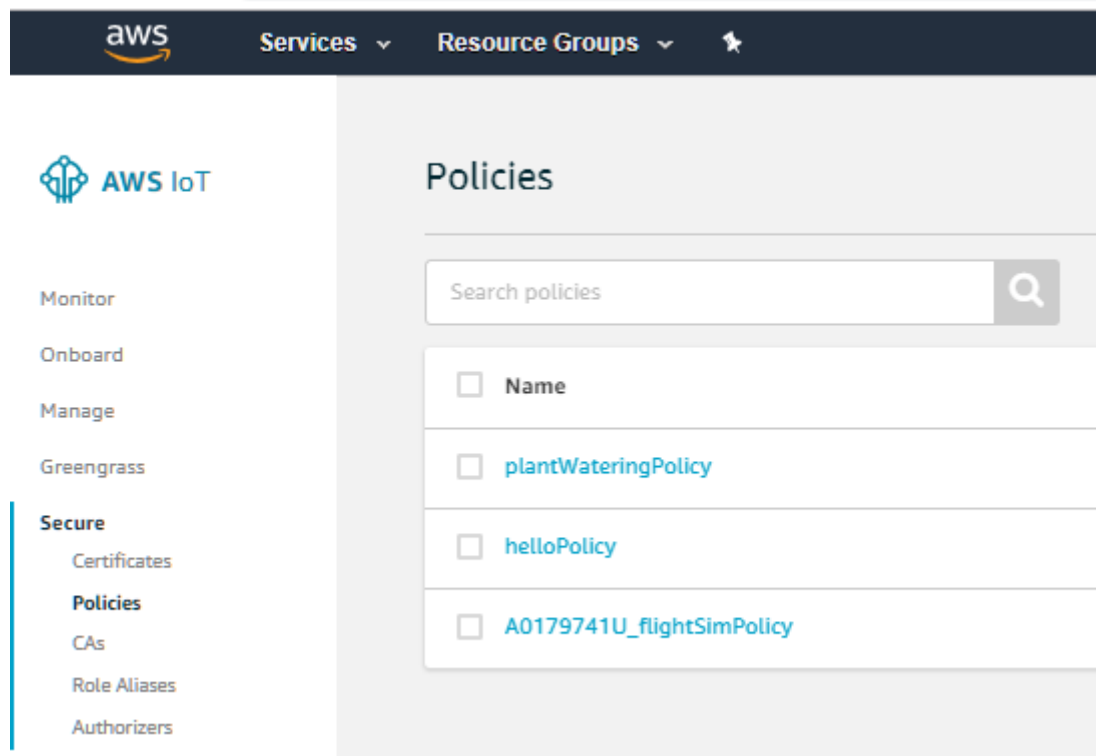


Figure 5: Setting up Policy part 1

Create a policy

Create a policy to define a set of authorized actions. You can authorize actions on one or more resources (things, topics, topic filters). To learn more about IoT policies go to the [AWS IoT Policies documentation page](#).

Name

EE5111_A0179741U_POLICY

Add statements

Policy statements define the types of actions that can be performed by a resource. **Advanced mode**

Action
iot:*

Resource ARN
*

Effect
<input checked="" type="checkbox"/> Allow <input type="checkbox"/> Deny

Remove

Add statement

Create

Figure 6: Setting up Policy part 2

Step 2: Create Thing for FD001

In this step, create a thing in AWS IoT to represent desktop/laptop as a device simulator. Devices connected to AWS IoT are represented by things in the AWS IoT registry. The registry keeps a record of all of the devices that are connected to the AWS account in AWS IoT. Under IoT Core console, select 'Manage' then 'Things' to 'Create' a Thing for FD001 as shown in Figure 7 and 8. Follow the red boxes guide and Enter a 'Name' and select 'next' as shown in Figure 9.

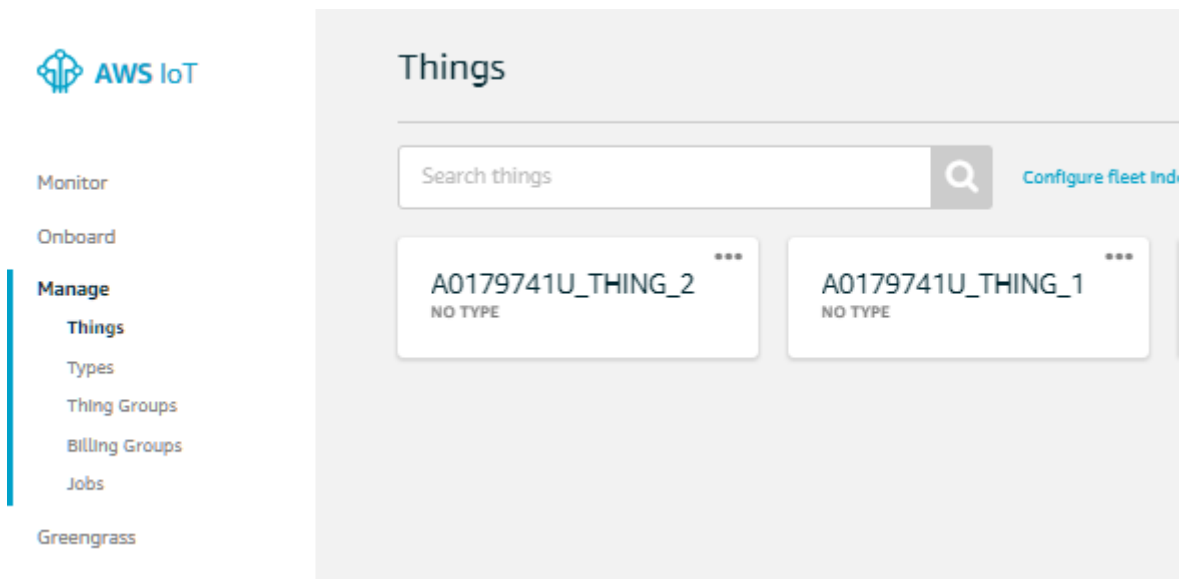


Figure 7: Setting up Thing part 1

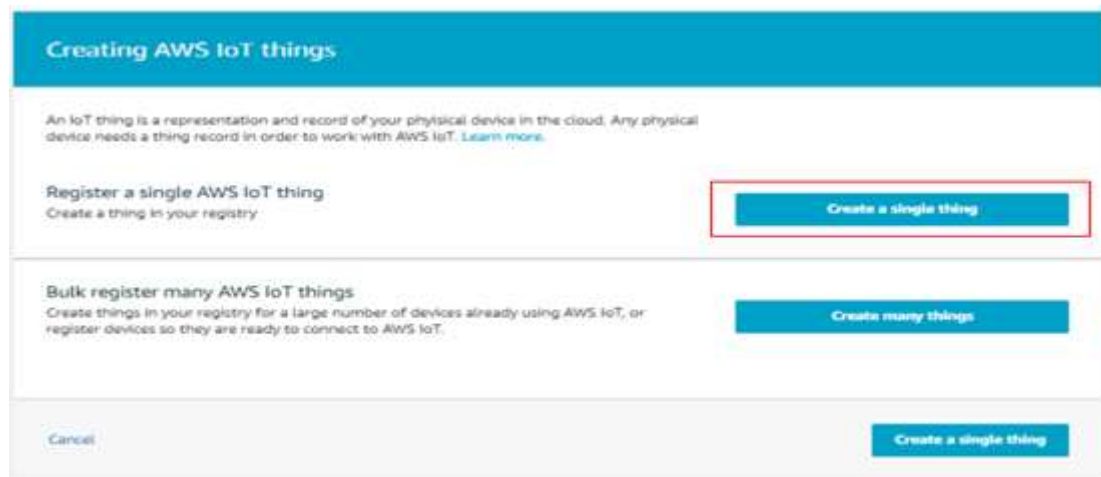


Figure 8: Setting up Thing part 2

CREATE A THING

STEP 1/5

Add your device to the thing registry

This step creates an entry in the thing registry and a thing shadow for your device.

Name

Apply a type to this thing

Using a thing type simplifies device management by providing consistent registry data for things that share a type. Types provide things with a common set of attributes, which describe the identity and capabilities of your device, and a description.

Thing Type

Add this thing to a group

Adding your thing to a group allows you to manage devices remotely using jobs.

Thing Group

Set searchable thing attributes (optional)

Enter a value for one or more of these attributes so that you can search for your things in the registry.

Attribute key	Value	
<input type="text" value="Provide an attribute key, e.g. Manufacturer"/>	<input type="text" value="Provide an attribute value, e.g. Acme-Corporation"/>	<input type="button" value="Clear"/>
<input type="button" value="Add another"/>		

Show thing shadow ▾

Figure 9:Setting up Thing part 3

Step 3: Create Certificate for FD001 Thing

Select 'Create certificate' for FD001 Thing to gain authentication to publish jet engine data to MQTT server for as shown in Figure 10. Download 'A certificate for this thing', 'A public key' and 'A private key' and select 'Activate' as shown in Figure 11. Put the 3 downloaded files with the Python file into the same folder. Click on the checkbox of the thing 'Name' to add policy to it as shown in Figure 12.

CREATE A THING

STOP 2/3

Add a certificate for your thing

A certificate is used to authenticate your device's connection to AWS IoT.

One-click certificate creation (recommended)
This will generate a certificate, public key, and private key using AWS IoT's certificate authority.

Create certificate

Create with CSR
Upload your own certificate signing request (CSR) based on a private key you own.

Create with CSR

Use my certificate
Register your CA certificate and use your own certificates for one or many devices.

Get started

Skip certificate and create thing
You will need to add a certificate to your thing later before your device can connect to AWS IoT.

Create thing without certificate

Figure 10: Setting up Thing part 4

Certificate created!

Download these files and save them in a safe place. Certificates can be retrieved at any time, but the private and public keys cannot be retrieved after you close this page.

In order to connect a device, you need to download the following:

A certificate for this thing	bfaedc01c3.cert.pem	Download
A public key	bfaedc01c3.public.key	Download
A private key	bfaedc01c3.private.key	Download

You also need to download a root CA for AWS IoT:
A root CA for AWS IoT [Download](#)

Activate

Cancel

Done

Attach a policy

Figure 11: Setting up Thing part 5

CREATE A THING

STEP 3/3

Add a policy for your thing

Select a policy to attach to this certificate:

Q Search policies

<input type="checkbox"/> plantWateringPolicy	View
<input type="checkbox"/> helloPolicy	View
<input type="checkbox"/> A0179741U_flightSimPolicy	View
<input checked="" type="checkbox"/> EE5111_A0179741U_POLICY	View

1 policy selected

Register Thing

Figure 12: Setting up Thing part 6

Step 4: Send and Receive Test Data for the Thing

In this step, we can send jet engines data to the thing shadow for the desktop/laptop as a device simulator. A thing's shadow is a JSON document, stored in AWS IoT , that AWS IoT uses to save and retrieve current state information for a device. The Device Shadow Service for AWS IoT maintains a shadow for each device connected to AWS IoT. Go to 'Things' and select 'Interact' as shown in Figure 13.

Things > EE5111_A0179741U_THING_1

THING

EE5111_A0179741U_THING_1

NO TYPE

Actions

Details

Security

Thing Groups

Billing Groups

Shadow

Interact

Activity

Jobs

Violations

Defender metrics

Thing ARN

Edit

A thing Amazon Resource Name uniquely identifies this thing.

arn:aws:iot:us-east-2:170968857749:thing/EE5111_A0179741U_THING_1

Type

Q No type

...

Figure 13: Thing's shadow send and receive data part 1

For MQTT, make a note of the value for each of the following MQTT topics, which enable you to set and get updates to the shadow as shown in Figure 14:

arn:aws:iot:us-east-2:170968857749:thing/EE5111_A0179741U_THING_1

\$aws/things/EE5111_A0179741U_THING_1/shadow/update

\$aws/things/EE5111_A0179741U_THING_1/shadow/update/accepted

\$aws/things/EE5111_A0179741U_THING_1/shadow/get

\$aws/things/EE5111_A0179741U_THING_1/shadow/get/accepted

Update to this thing shadow

\$aws/things/EE5111_A0179741U_THING_1/shadow/update

Update to this thing shadow was accepted

\$aws/things/EE5111_A0179741U_THING_1/shadow/update/accepted

Update this thing shadow documents

\$aws/things/EE5111_A0179741U_THING_1/shadow/update/documents

Update to this thing shadow was rejected

\$aws/things/EE5111_A0179741U_THING_1/shadow/update/rejected

Get this thing shadow

\$aws/things/EE5111_A0179741U_THING_1/shadow/get

Get this thing shadow accepted

\$aws/things/EE5111_A0179741U_THING_1/shadow/get/accepted

Figure 14: Things's shadow send and receive data part 2

For Subscription topic, enter the MQTT topic value in Figure 13 of this procedure for Update to thing shadow (for example, \$aws/things/EE5111_A0179741U_THING_1/shadow/update), and then choose Subscribe to topic. Repeat the same procedure for the MQTT topic values for (for example, \$aws/things/EE5111_A0179741U_THING_1/shadow/get) and (for example, \$aws/things/EE5111_A0179741U_THING_1/shadow/get/accepted) as shown in Figure 15.

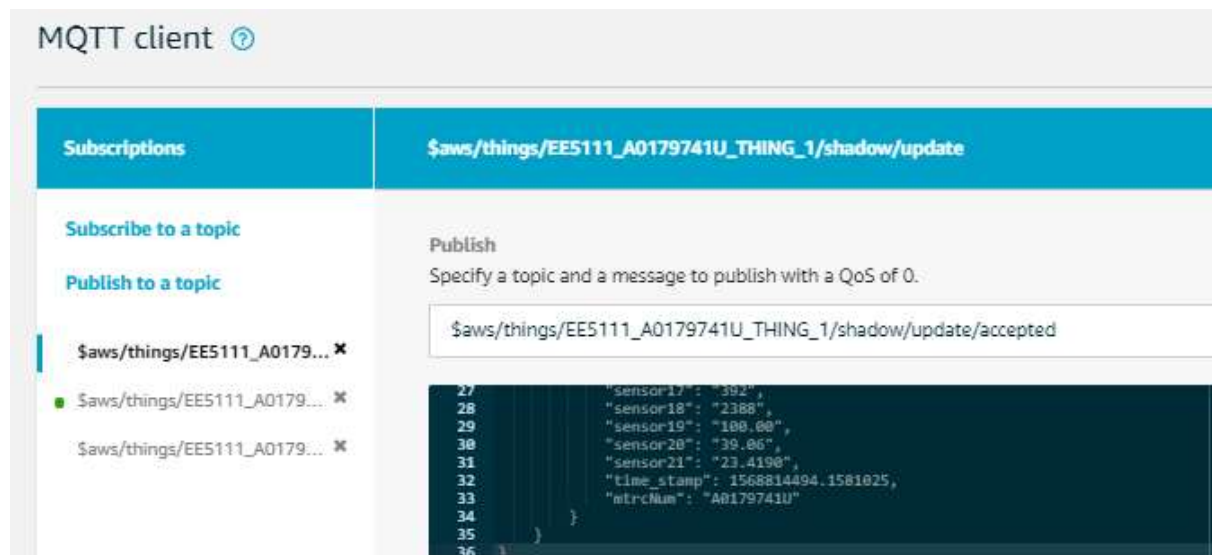


Figure 15: Things's shadow send and receive data part 3

To get that data from the shadow, choose the MQTT topic value for Get this thing shadow (for example, \$aws/things/EE5111_A0179741U_THING_1/shadow/get). In the message payload area, replace the current payload with the following payload and 'Publish to topic' as shown in Figure 16.

Last update: Sep 21, 2019 10:42:26 AM +0800

Shadow state:

```
{
  "reported": {
    "id": "FD001_1",
    "cycle": "1",
    "os1": "-0.0007",
    "os2": "-0.0004",
    "os3": "100.0",
    "sensor1": "518.67",
    "sensor2": "641.82",
    "sensor3": "1589.70",
    "sensor4": "1400.60",
    "sensor5": "14.62",
    "sensor6": "21.61",
    "sensor7": "554.36",
    "sensor8": "2388.06",
    "sensor9": "9046.19",
    "sensor10": "1.30",
    "sensor11": "47.47",
    "sensor12": "521.66",
    "sensor13": "2388.02",
    "sensor14": "8138.62",
    "sensor15": "8.4195",
    "sensor16": "0.03",
    "sensor17": "392",
    "sensor18": "2388",
    "sensor19": "100.00",
    "sensor20": "39.06",
    "sensor21": "23.4190",
    "time_stamp": 1568814494.1581025,
    "mtrcNum": "A0179741U"
  }
}
```

Figure 16: JSON document

Step 5: Creating an Amazon DynamoDB Rule

DynamoDB rules allow users to take information from an incoming MQTT message (When running the Python script) and write it to a DynamoDB table. In the AWS IoT console, in the navigation panel, choose 'Act' and select 'Create' as shown in Figure 18.

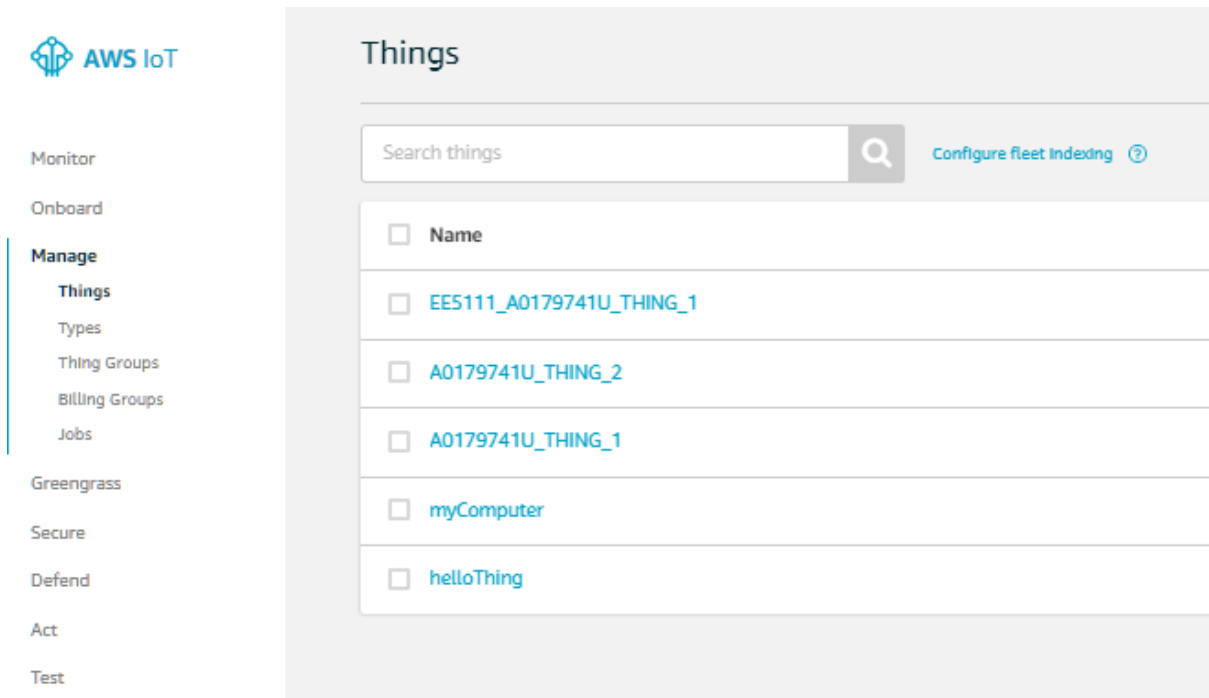


Figure 17: Amazon DynamoDB part 1

On the Create a rule page, enter a name, description for your rule and select 'Add action' to choose 'Split message into multiple columns of a DynamoDB table (DynamoDBv2)', and then choose 'Configure action' as shown in Figure 19, 20, 21 and 22.

The screenshot shows the 'Create a rule' page in the AWS IoT console. The page has a blue header with the title 'Create a rule'. Below the header, there is a paragraph of text: 'Create a rule to evaluate messages sent by your things and specify what to do when a message is received (for example, write data to a DynamoDB table or invoke a Lambda function)'. There are two input fields: 'Name' with the value 'EE5111_A0179741U_DYNDB_RULE' and 'Description' with the value 'DynamoDB rule.'. Below these fields is a section titled 'Rule query statement' with the text 'Indicate the source of the messages you want to process with this rule.' and a dropdown menu labeled 'Using SQL version' with the value '2016-03-23'.

Figure 18: Amazon DynamoDB part 2

Rule query statement

Indicate the source of the messages you want to process with this rule.

Using SQL version

2016-03-23

Rule query statement

SELECT <Attribute> FROM <Topic Filter> WHERE <Condition>. For example: SELECT temperature FROM 'iot/topic' WHERE temperature > 50. To learn more, see [AWS IoT SQL Reference](#).

```
1 SELECT * FROM 'aws/things/+shadow/update'
2
```

Set one or more actions

Select one or more actions to happen when the above rule is matched by an inbound message. Actions define additional activities that occur when messages arrive, like storing them in a database, invoking cloud functions, or sending notifications. (*.required)

Add action




Figure 19: Amazon DynamoDB part 3

Select an action

Select an action.

- ☐  Insert a message into a DynamoDB table
DYNAMODB
- ☒  Split message into multiple columns of a DynamoDB table (DynamoDBv2)
DYNAMODBv2

Figure 20: Amazon DynamoDB part 4

- ☐  Send a message to a Salesforce IoT input stream
SALESFORCE IOT
- ☐  Send a message to IoT Analytics
IOT ANALYTICS
- ☐  Send a message to an IoT Events Input
IOT EVENTS
- ☐  Start a Step Functions state machine execution
STEP FUNCTIONS

Cancel

Configure action

Figure 21: Amazon DynamoDB part 5

On the Configure action page, choose 'Create a new resource'. This action will bring you DynamoDB table as shown in Figure 23.

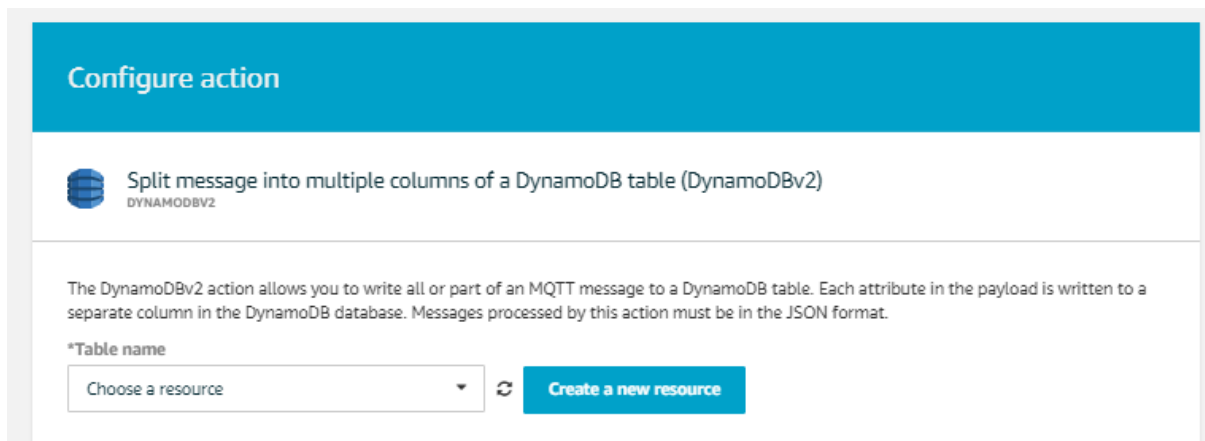


Figure 22: Amazon DynamoDB part 7

On the Amazon DynamoDB page, choose 'Create table' as shown in Figure 24.

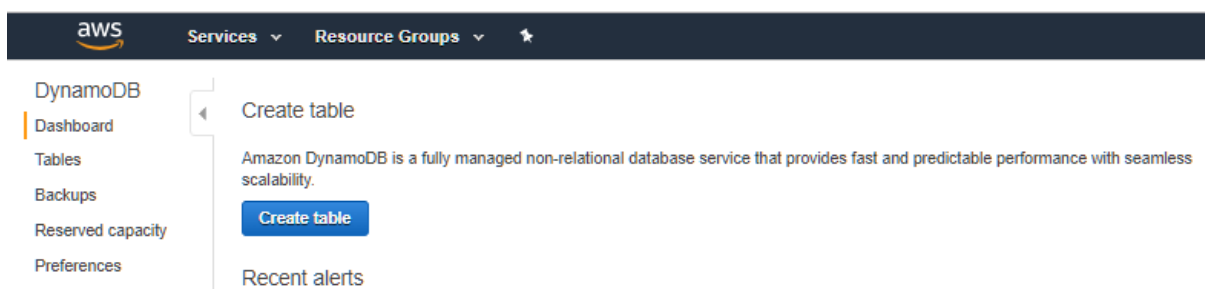


Figure 23: Amazon DynamoDB part 8

On the Create DynamoDB table page, enter a 'Name' in Table name. In Partition key, enter 'id'. Select Add sort key, and then enter 'timestamp' in the Sort key field. Row represents an id of the sensors for jet engine. Timestamp represents the time in UTC (e.g. UTC 2019-01-28 14:41:15.237). Choose String for both the partition and sort keys, and then choose Create. It takes a few seconds to create your DynamoDB table. Close the browser tab where the Amazon DynamoDB console is open. If you don't close the tab, your DynamoDB table is not displayed in the Table name list on the Configure action page of the AWS IoT console.

Create DynamoDB table

Tutorial ?

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

Table name* ⓘ

Primary key* Partition key

ⓘ

☒ Add sort key

ⓘ

Table settings

Default settings provide the fastest way to get started with your table. You can modify these default settings now or after your table has been created.

- ☒ Use default settings
- No secondary indexes.
 - Provisioned capacity set to 5 reads and 5 writes.
 - Basic alarms with 80% upper threshold using SNS topic "dynamodb".
 - Encryption at Rest with DEFAULT encryption type.

ⓘ You do not have the required role to enable Auto Scaling by default.
Please refer to [documentation](#).

+ Add tags **NEW!**

Additional charges may apply if you exceed the AWS Free Tier levels for CloudWatch or Simple Notification Service. Advanced alarm settings are available in the CloudWatch management console.

Cancel **Create**

Figure 24: Amazon DynamoDB part 9

On the Amazon DynamoDB page, choose 'Manage Stream' and select 'Enable' for DynamoDB table to receive data as shown in Figure 26 and 27.

EE5111_A0179741U_DDB_TABLE [Close](#)

Overview Items Metrics Alarms Capacity Indexes Global Tables Backups Tri

Recent alerts

No CloudWatch alarms have been triggered for this table.

Stream details

Stream enabled	No
View type	-
Latest stream ARN	-

Manage Stream

Figure 25: Amazon DynamoDB part 10

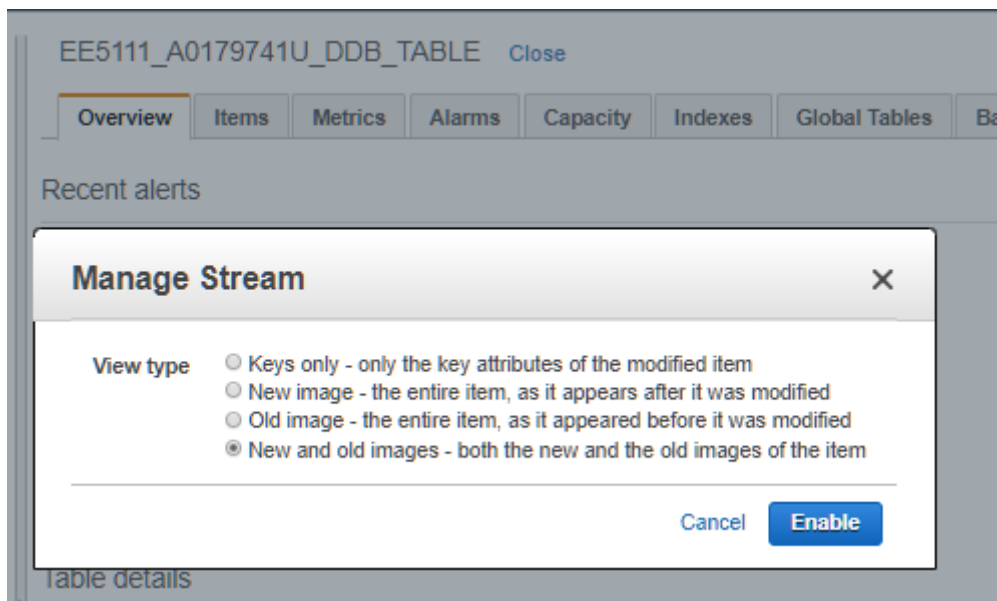


Figure 26: Amazon DynamoDB part 11

Back on the Configure action page, choose the table created earlier from the Table name list. Select 'Create Role' and attach Policy for the table as shown in Figure 28 and 29.

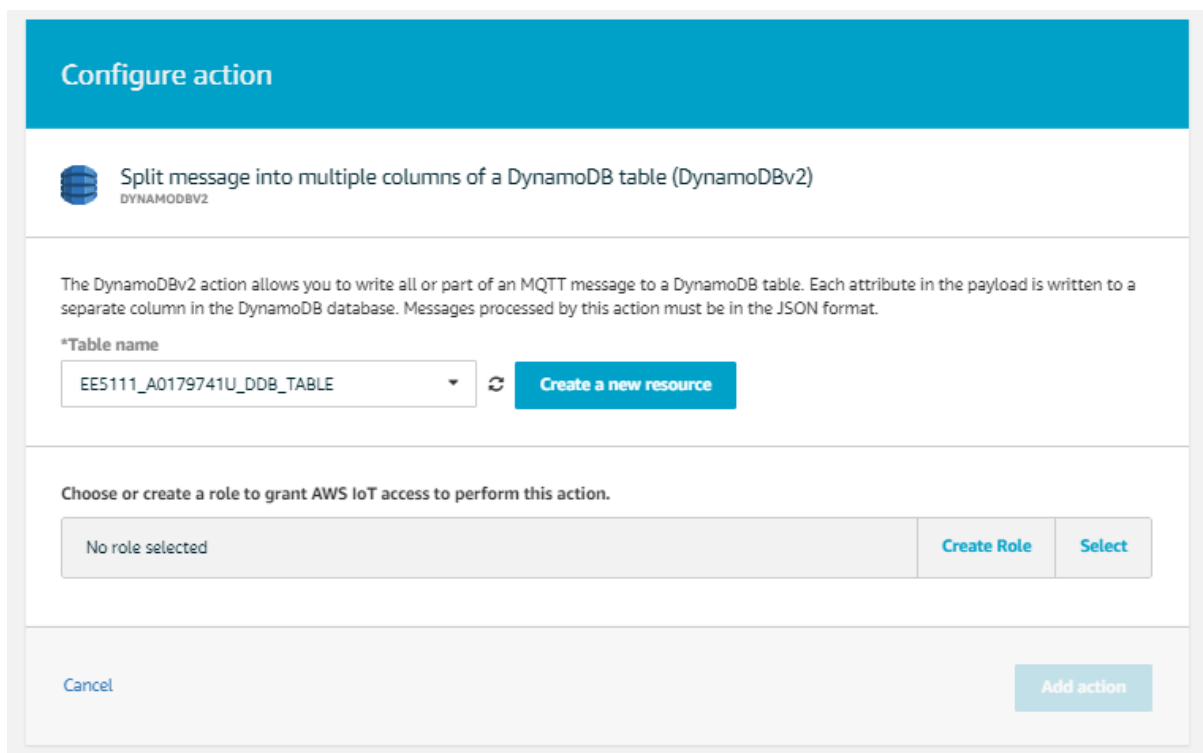


Figure 27: Amazon DynamoDB part 12

Choose or create a role to grant AWS IoT access to perform this action.

EE5111_A0179741U_DDB_ROLE	Policy Attached ✓	Create Role	Select
---------------------------	-------------------	-------------	--------

Cancel Add action

Figure 28: Amazon DynamoDB part 13

On the Configure action page, enter the JSON message: `SELECT state.reported.* FROM '$aws/things/EE5111_A0179741U_THING_1/shadow/update/accepted'` for rule query statement in the window to retrieve data then select 'Create Rule' as shown in Figure 30 and 31.

Rule query statement
Indicate the source of the messages you want to process with this rule.


Using SQL version

2016-03-23 ▼

Rule query statement
SELECT <Attribute> FROM <Topic Filter> WHERE <Condition>. For example: SELECT temperature FROM 'iot/topic' WHERE temperature > 50. To learn more, see [AWS IoT SQL Reference](#).

```
1 SELECT * FROM '$aws/things/+shadow/update'
```

Set one or more actions
Select one or more actions to happen when the above rule is matched by an inbound message. Actions define additional activities that occur when messages arrive, like storing them in a database, invoking cloud functions, or sending notifications. (*.required)



Split message into multiple columns of a DynamoDB table...
EE5111_A0179741U_DDB_TABLE

Remove Edit ►

Figure 29: Amazon DynamoDB part 14

Error action
 Optionally set an action that will be executed when something goes wrong with processing your rule.

[Add action](#)

Tags
 Apply tags to your resources to help organize and identify them. A tag consists of a case-sensitive key-value pair. [Learn more](#) about tagging your AWS resources.

Tag name

Value

[Clear](#)

[Add another](#)

[Cancel](#) [Create rule](#)

Figure 30: Amazon DynamoDB part 15

Step 6: Creating Python codes

From the given AWS given skeleton codes, modify the following for Thing1.

```
SHADOW_CLIENT = "EE5111_A0179741U_THING_1"
```

```
HOST_NAME = "a12d228r28ffvb-ats.iot.us-east-2.amazonaws.com"
```

```
ROOT_CA = "AmazonRootCA1.pem.txt"
```

```
PRIVATE_KEY = "bfaedc01c3-private.pem.key"
```

```
CERT_FILE = "bfaedc01c3-certificate.pem.crt"
```

```
SHADOW_HANDLER = "EE5111_A0179741U_THING_1"
```

The codes are able to read and publish data from trainFD001.txt to your thing under AWS IoT platform at the rate of 10 seconds per row. Overwrite column 'id' of the engine as 'FD001' + id (e.g. FD001_12), add one more columns 'timestamp' as timestamps in UTC (e.g. UTC 2019-01-28 14:41:15.237), add one more column that containing Matric number, concatenate data into JSON string and send it to Thing1 shadow handler as shown with comments in Figure 32, 33 and 34. Full codes of FD001 Thing1 and FD002 Thing2 are in the appendix of this report.

```

from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTShadowClient
import random, time, json, enum

class dataDescription(enum.Enum):
    ....id.....=0
    ....cycle.....=1
    ....os1.....=2
    ....os2.....=3
    ....os3.....=4
    ....sensor1.....=5
    ....sensor2.....=6
    ....sensor3.....=7
    ....sensor4.....=8
    ....sensor5.....=9
    ....sensor6.....=10
    ....sensor7.....=11
    ....sensor8.....=12
    ....sensor9.....=13
    ....sensor10.....=14
    ....sensor11.....=15
    ....sensor12.....=16
    ....sensor13.....=17
    ....sensor14.....=18
    ....sensor15.....=19
    ....sensor16.....=20
    ....sensor17.....=21
    ....sensor18.....=22
    ....sensor19.....=23
    ....sensor20.....=24
    ....sensor21.....=25
    ....lineLength.....=26
    ....time_stamp.....=27
    ....matric_num.....=28

```

Figure 31: Python code part 1

```

data_dict = {
    'id': None,
    'cycle': None,
    'os1': None,
    'os2': None,
    'os3': None,
    'sensor1': None,
    'sensor2': None,
    'sensor3': None,
    'sensor4': None,
    'sensor5': None,
    'sensor6': None,
    'sensor7': None,
    'sensor8': None,
    'sensor9': None,
    'sensor10': None,
    'sensor11': None,
    'sensor12': None,
    'sensor13': None,
    'sensor14': None,
    'sensor15': None,
    'sensor16': None,
    'sensor17': None,
    'sensor18': None,
    'sensor19': None,
    'sensor20': None,
    'sensor21': None,
    'time_stamp': None,
    'mtrcNum': None
}

#open a file as read-only
rdfile = open('train_FD001.txt')

# A random programmatic shadow client ID.
SHADOW_CLIENT = "EE5111_A0179741U_THING_1"

```

Figure 32: Python code part 2

```

# The unique hostname that &IoT; generated for
# this device.
# ThisHasToBeUpdated
HOST_NAME = "a12d228r28ffvb-ats.iot.us-east-2.amazonaws.com"

# The relative path to the correct root CA file for &IoT;;
# which you have already saved onto this device.
# ThisHasToBeUpdated
ROOT_CA = "AmazonRootCA1.pem.txt"

# The relative path to your private key file that
# &IoT; generated for this device, which you
# have already saved onto this device.
# ThisHasToBeUpdated
PRIVATE_KEY = "bfaedc0lc3-private.pem.key"

# The relative path to your certificate file that
# &IoT; generated for this device, which you
# have already saved onto this device.
# ThisHasToBeUpdated
CERT_FILE = "bfaedc0lc3-certificate.pem.crt"

# A programmatic shadow handler name prefix.
# ThisHasToBeUpdated
SHADOW_HANDLER = "EE5111_A0179741U_THING_1"

# Automatically called whenever the shadow is updated.
def myShadowUpdateCallback(payload, responseStatus, token):
    print()
    print('UPDATE: $aws/things/' + SHADOW_HANDLER +
          '/shadow/update/#')
    print("payload = " + payload)
    print("responseStatus = " + responseStatus)
    print("token = " + token)

```

Figure 33: Python code part 3

```

# Create, configure, and connect a shadow client.
myShadowClient = AWSIoTMQTTShadowClient(SHADOW_CLIENT)
myShadowClient.configureEndpoint(HOST_NAME, 8883)
myShadowClient.configureCredentials(ROOT_CA, PRIVATE_KEY,
    CERT_FILE)
myShadowClient.configureConnectDisconnectTimeout(10)
myShadowClient.configureMQTTOperationTimeout(5)
myShadowClient.connect()

# Create a programmatic representation of the shadow.
myDeviceShadow = myShadowClient.createShadowHandlerWithName(
    SHADOW_HANDLER, True)

# Keep generating random test data until this script
# stops running.
# To stop running this script, press Ctrl+C.

```

Figure 34: Python code part 3


```

line = rdfile.readline()
while line:
    print(line)
    line = line.split()
    ....
    #override id value as FD001_<id>
    data_dict['id'] = 'FD001_' + line[dataDescription.id.value]
    #add time_stamp
    data_dict['time_stamp'] = time.time()
    #add matrix number
    data_dict['mtrcNum'] = 'A0179741U'
    #read rest of the entries from line and add them to dict
    if len(line) != dataDescription.lineLength.value:
        print('ERROR: This line does not have enough data.')
        exit()
    data_dict['cycle'] = line[dataDescription.cycle.value]
    data_dict['os1'] = line[dataDescription.os1.value]
    data_dict['os2'] = line[dataDescription.os2.value]
    data_dict['os3'] = line[dataDescription.os3.value]
    data_dict['sensor1'] = line[dataDescription.sensor1.value]
    data_dict['sensor2'] = line[dataDescription.sensor2.value]
    data_dict['sensor3'] = line[dataDescription.sensor3.value]
    data_dict['sensor4'] = line[dataDescription.sensor4.value]
    data_dict['sensor5'] = line[dataDescription.sensor5.value]
    data_dict['sensor6'] = line[dataDescription.sensor6.value]
    data_dict['sensor7'] = line[dataDescription.sensor7.value]
    data_dict['sensor8'] = line[dataDescription.sensor8.value]
    data_dict['sensor9'] = line[dataDescription.sensor9.value]
    data_dict['sensor10'] = line[dataDescription.sensor10.value]
    data_dict['sensor11'] = line[dataDescription.sensor11.value]
    data_dict['sensor12'] = line[dataDescription.sensor12.value]
    data_dict['sensor13'] = line[dataDescription.sensor13.value]
    data_dict['sensor14'] = line[dataDescription.sensor14.value]
    data_dict['sensor15'] = line[dataDescription.sensor15.value]
    data_dict['sensor16'] = line[dataDescription.sensor16.value]
    data_dict['sensor17'] = line[dataDescription.sensor17.value]
    data_dict['sensor18'] = line[dataDescription.sensor18.value]
    data_dict['sensor19'] = line[dataDescription.sensor19.value]
    data_dict['sensor20'] = line[dataDescription.sensor20.value]
    data_dict['sensor21'] = line[dataDescription.sensor21.value]
    data_json = json.dumps(data_dict)
    print(data_json + "\n")
    #myDeviceShadow.shadowUpdate(data_json, myShadowUpdateCallback, 5)
    myDeviceShadow.shadowUpdate(
        '{"state":{"reported":"' + data_json + '}}',
        myShadowUpdateCallback, 5)
    line = rdfile.readline()
    time.sleep(10)

```

Figure 35: Python code part 3

Step 7: Executing Python codes and streaming data to AWS DynamoDB table

Run the Python codes and shown in Figure 35 and an update will appear for every 10 seconds interval on Python Shell upon successful streaming of the FD001 engine data to AWS DynamoDB table. To view the published data, go to AWS DynamoDB table and select 'Items' as shown in Figure 36.

```
(base) C:\Users\Ashfaq\Desktop\aws\TASK_1\AWS_IOT\AB179741U_AWS_IOT\THING1>python aws_iot.py
1 1 -0.0007 -0.0004 100.0 518.67 641.82 1589.70 1400.60 14.02 21.01 554.36 2388.00 9046.10 1.38 47.47 521.86 2388.02 8138.62 0.4195 0.03 392 2388 100.00 30.00 23.4190

{"id": "FD001_1", "cycle": "1", "os1": "-0.0007", "os2": "-0.0004", "os3": "100.0", "sensor1": "518.67", "sensor2": "641.82", "sensor3": "1589.70", "sensor4": "1400.60",
"sensor5": "14.02", "sensor6": "21.01", "sensor7": "554.36", "sensor8": "2388.00", "sensor9": "9046.10", "sensor10": "1.38", "sensor11": "47.47", "sensor12": "521.86",
"sensor13": "2388.02", "sensor14": "8138.62", "sensor15": "0.4195", "sensor16": "0.03", "sensor17": "392", "sensor18": "2388", "sensor19": "100.00", "sensor20": "30.00", "sensor21": "23.4190",
"time_stamp": "1569065320.8806958", "mtrchum": "AB179741U"}

UPDATE: $aws/things/EE5111_A0179741U_THING1/shadow/update/#
payload = {"state":{"reported":{"id":"FD001_1","cycle":"1","os1":-0.0007,"os2":-0.0004,"os3":100.0,"sensor1":518.67,"sensor2":641.82,"sensor3":1589.70,"sensor4":1400.60,"sensor5":14.02,"sensor6":21.01,"sensor7":554.36,"sensor8":2388.00,"sensor9":9046.10,"sensor10":1.38,"sensor11":47.47,"sensor12":521.86,"sensor13":2388.02,"sensor14":8138.62,"sensor15":0.4195,"sensor16":0.03,"sensor17":392,"sensor18":2388,"sensor19":100.00,"sensor20":30.00,"sensor21":23.4190,"time_stamp":1569065320.8806958,"mtrchum":AB179741U}}, "metadata":{"reported":{"id":{"timestamp":1569065323},"cycle":{"timestamp":1569065323},"os1":{"timestamp":1569065323},"os2":{"timestamp":1569065323},"os3":{"timestamp":1569065323},"sensor1":{"timestamp":1569065323},"sensor2":{"timestamp":1569065323},"sensor3":{"timestamp":1569065323},"sensor4":{"timestamp":1569065323},"sensor5":{"timestamp":1569065323},"sensor6":{"timestamp":1569065323},"sensor7":{"timestamp":1569065323},"sensor8":{"timestamp":1569065323},"sensor9":{"timestamp":1569065323},"sensor10":{"timestamp":1569065323},"sensor11":{"timestamp":1569065323},"sensor12":{"timestamp":1569065323},"sensor13":{"timestamp":1569065323},"sensor14":{"timestamp":1569065323},"sensor15":{"timestamp":1569065323},"sensor16":{"timestamp":1569065323},"sensor17":{"timestamp":1569065323},"sensor18":{"timestamp":1569065323},"sensor19":{"timestamp":1569065323},"sensor20":{"timestamp":1569065323},"sensor21":{"timestamp":1569065323},"time_stamp":{"timestamp":1569065323},"mtrchum":{"timestamp":1569065323}}},"version":2,"timestamp":1569065323,"clientToken":"12183f21-7dca-4c31-9d6a-273fed8e53ee"}
responseStatus = accepted
token = 12183f21-7dca-4c31-9d6a-273fed8e53ee
```

Figure 36: Executing Python for Thing1

Scan: [Table] EE5111_A0179741U_ODB_TABLE: id, time_stamp

Viewing 1 to 4 items

id	time_stamp	clientToken	cycle	mtrchum	os1	os2	os3
FD001_1	1569064863.4585123	ac5d10cc-5f0b-431d-9293-a6b70f319111	1	A0179741U	-0.0007	-0.0004	100.0
FD001_1	1569064675.9074966	51f7ed2b-c0f1-4652-b32f-d27f6abe7909	2	A0179741U	0.0019	-0.0003	100.0
FD001_1	1569064606.0004096	deab1435-3c85-4ede-b896-c0b140ce9ea5	3	A0179741U	-0.0043	0.0003	100.0
FD001_1	1569064606.0118282	5d95ab66-beeb-48e1-a606-443d9e9817ad	4	A0179741U	0.0007	0.0006	100.0

Figure 37: Updated DynamoDb Table

Create Thing2 for FD002

To simulate two IoT things now, add one more thing under AWS IoT platform and one more certificate, create a copy of the Jupyter notebook above, renaming the client name, certificate, data source train_FD002.txt, modify the rule under AWS IoT platform to be triggered by '\$aws/things/+shadow/update/accepted' as shown in Thing1 for FD001 step 1–6. Now, this rule will push data of both engines from both things. Let the two Jupyter notebooks run at the same time as shown in Figure 37, simulating the two “things” to run in parallel to publish data with a sampling rate of one record per 10 seconds in each thing.

```
Anaconda Prompt (Anaconda3) - python aws_iot.py
7.93 0.0336 0.03 308 2319 100.0

{"id": "FD002_4", "cycle": "8", "os1": "0.0007", "os2": "0.0004", "os3": "100.0", "sensor1": "518.67", "sensor2": "641.82", "sensor3": "1589.70", "sensor4": "1400.60",
"sensor5": "14.02", "sensor6": "21.01", "sensor7": "554.36", "sensor8": "2388.00", "sensor9": "9046.10", "sensor10": "1.38", "sensor11": "47.47", "sensor12": "521.86",
"sensor13": "2388.02", "sensor14": "8138.62", "sensor15": "0.4195", "sensor16": "0.03", "sensor17": "392", "sensor18": "2388", "sensor19": "100.00", "sensor20": "30.00", "sensor21": "23.4190",
"time_stamp": "1569065320.8806958", "mtrchum": "AB179741U"}

UPDATE: $aws/things/EE5111_A0179741U_THING1/shadow/update/#
payload = {"code":400,"message":"Missing required node: state","clientToken":"d803dbb7-8b03-4a5b-8aa6-128e7446f795"}
responseStatus = rejected
token = d803dbb7-8b03-4a5b-8aa6-128e7446f795

Anaconda Prompt (Anaconda3) - python aws_iot.py
7.93 0.0336 0.03 308 2319 100.0

{"id": "FD002_4", "cycle": "8", "os1": "0.0007", "os2": "0.0004", "os3": "100.0", "sensor1": "518.67", "sensor2": "641.82", "sensor3": "1589.70", "sensor4": "1400.60",
"sensor5": "14.02", "sensor6": "21.01", "sensor7": "554.36", "sensor8": "2388.00", "sensor9": "9046.10", "sensor10": "1.38", "sensor11": "47.47", "sensor12": "521.86",
"sensor13": "2388.02", "sensor14": "8138.62", "sensor15": "0.4195", "sensor16": "0.03", "sensor17": "392", "sensor18": "2388", "sensor19": "100.00", "sensor20": "30.00", "sensor21": "23.4190",
"time_stamp": "1569065320.8806958", "mtrchum": "AB179741U"}

UPDATE: $aws/things/AB179741U_THING2/shadow/update/#
payload = {"code":400,"message":"Missing required node: state","clientToken":"d803dbb7-8b03-4a5b-8aa6-128e7446f795"}
responseStatus = rejected
token = d803dbb7-8b03-4a5b-8aa6-128e7446f795
```

Figure 38: Executing Python for Thing1 and Thing2

To view both jet engines data for Thing1 and Thing2, go to Amazon DynamoDB page select 'Items', click on 'Refresh' and click on 'timestamp' to view FD001 and FD002 data as shown in Figure 38.

id	time_stamp	clientToken	cycle	metricNum	oa1	oa2	oa3
FD002_1	1569066271.363368	aae6507e-44a8-4842-b3c0-4b442d588e70	1	A0179741U	34.9983	0.8400	100.0
FD002_1	1569066083.958577	ad569f0e-2110-4717-a081-58c785d05a3	2	A0179741U	41.9982	0.8408	100.0
FD002_1	1569066093.976290	edd6bd5e-7961-4872-9398-d148c4c26cac	3	A0179741U	24.9988	0.8218	90.0
FD002_1	1569066104.005574	5b98ae2f-dca2-4c54-bcb-4650a847e530	4	A0179741U	42.0077	0.8418	100.0
FD002_1	1569066114.030171	452187a8-8ca5-4755-8de6-acbd5abff313	5	A0179741U	25.0005	0.8293	90.0
FD002_1	1569066124.056468	138a9517-fb8f-4405-95d9-9768b0f90ee	6	A0179741U	25.0045	0.8295	90.0
FD001_1	1569064663.458512	ac9d18cc-5f0b-431d-8293-a8b70f31911	1	A0179741U	-0.0007	-0.0004	100.0
FD001_1	1569064675.987496	51f7ed2b-c0f1-4852-b32f-d276eab79f9	2	A0179741U	0.0019	-0.0003	100.0
FD001_1	1569064686.000408	0eab1435-3c88-4ede-b599-c0fd4ece9ea5	3	A0179741U	-0.0043	0.0003	100.0
FD001_1	1569064686.0118282	5d95ab66-beeb-48e1-a606-f43d9e9817ad	4	A0179741U	0.0007	0.0000	100.0

Figure 39: AWS DynamoDB Table for Thing1 and Thing2

Data visualization on Redash for both Thing1 and Thing2

IAM Policies

Using Identity-Based Policies (IAM Policies) for Amazon DynamoDB allows an account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles) and thereby grant permissions to perform operations on Amazon DynamoDB resources. From IoT console interface, search for 'IAM', select 'Users' and click on 'Add user' as shown in Figure 39.

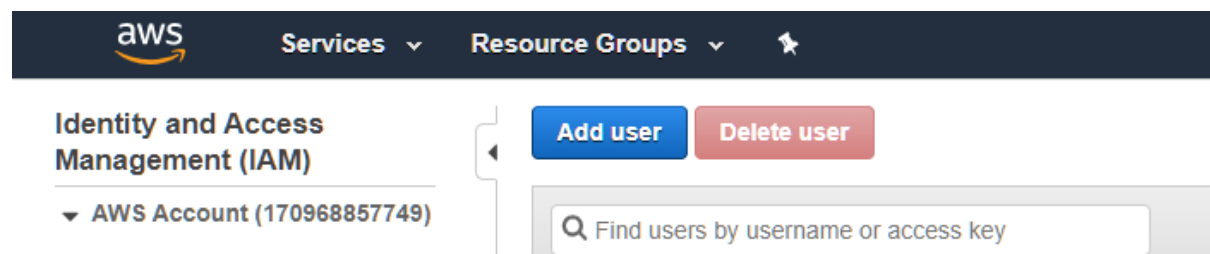


Figure 40: AWS IAM Table for Things part 1

Enter 'User name', check the checkboxes for 'Programmatic access' and 'AWS Management Console access' and select 'Next Permission' as shown in Figure 40.

Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name*

[+ Add another user](#)

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

Access type* ☒ **Programmatic access**
Enables an access key ID and secret access key for the AWS API, CLI, SDK, and other development tools.

☒ **AWS Management Console access**
Enables a password that allows users to sign-in to the AWS Management Console.

Console password* ☒ Autogenerated password
☐ Custom password

Require password reset ☒ User must create a new password at next sign-in
Users automatically get the [IAMUserChangePassword](#) policy to allow them to change their own password.

* Required

[Cancel](#)

[Next: Permissions](#)


Figure 41: AWS IAM Table for Things part 2


Select 'Attach existing policies directly' and search for 'AmazonDynamo Db' under Filter policies. Check the 3 checkboxes, click on 'Next Tags', 'Next Review' and 'Create user' as shown in Figure 41, 42 and 43.


Add user

1 2 3 4 5

▼ Set permissions

 Add user to group

 Copy permissions from existing user

 Attach existing policies directly

[Create policy](#)








Filter policies ▼		Q AmazonDynamo Db			Showing 16 results
	Policy name ▼	Type	Used as	Description	
<input checked="" type="checkbox"/>	 AmazonDocDBCon...	AWS managed	Permissions policy (1)	Provides full access to manage Amazon ...	
<input checked="" type="checkbox"/>	 AmazonDocDBFull...	AWS managed	Permissions policy (1)	Provides full access to Amazon Docume...	
<input checked="" type="checkbox"/>	 AmazonDocDBRea...	AWS managed	Permissions policy (1)	Provides read-only access to Amazon Do...	
<input checked="" type="checkbox"/>	 AmazonDynamoDB...	AWS managed	Permissions policy (1)	Provides full access to Amazon Dynamo...	
<input checked="" type="checkbox"/>	 AmazonDynamoDB...	AWS managed	Permissions policy (1)	Provides full access to Amazon Dynamo...	

Figure 42: AWS IAM Table for Things part 3

Add user

1 2 3 4 5

Add tags (optional)

IAM tags are key-value pairs you can add to your user. Tags can include user information, such as an email address, or can be descriptive, such as a job title. You can use the tags to organize, track, or control access for this user. [Learn more](#)

Key	Value (optional)	Remove
<input type="text" value="Add new key"/>	<input type="text"/>	

You can add 50 more tags.

Cancel Previous Next: Review

Figure 43: AWS IAM Table for Things part 4

Add user

1 2 3 4 5

Review

Review your choices. After you create the user, you can view and download the autogenerated password and access key.

User details

User name	EE5111_A0179741U
AWS access type	Programmatic access and AWS Management Console access
Console password type	Autogenerated
Require password reset	Yes
Permissions boundary	Permissions boundary is not set

Permissions summary

The following policies will be attached to the user shown above.

Type	Name
Managed policy	AmazonDocDBConsoleFullAccess
Managed policy	AmazonDocDBFullAccess
Managed policy	AmazonDocDBReadOnlyAccess
Managed policy	AmazonDynamoDBFullAccess
Managed policy	AmazonDynamoDBFullAccesswithDataPipeline
Managed policy	IAMUserChangePassword

Tags

Cancel Previous Create user

Figure 44: AWS IAM Table for Things part 5

Download the 'credential' file containing User name, Password, Access key ID, Secret access key and Console login link by clicking on 'Download .csv' as shown in Figure 44. The authentication access information will be used later on for data visualization on Redash platform.

Add user

1

2

3

4

5

✓

Success
You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: <https://170968857749.signin.aws.amazon.com/console>

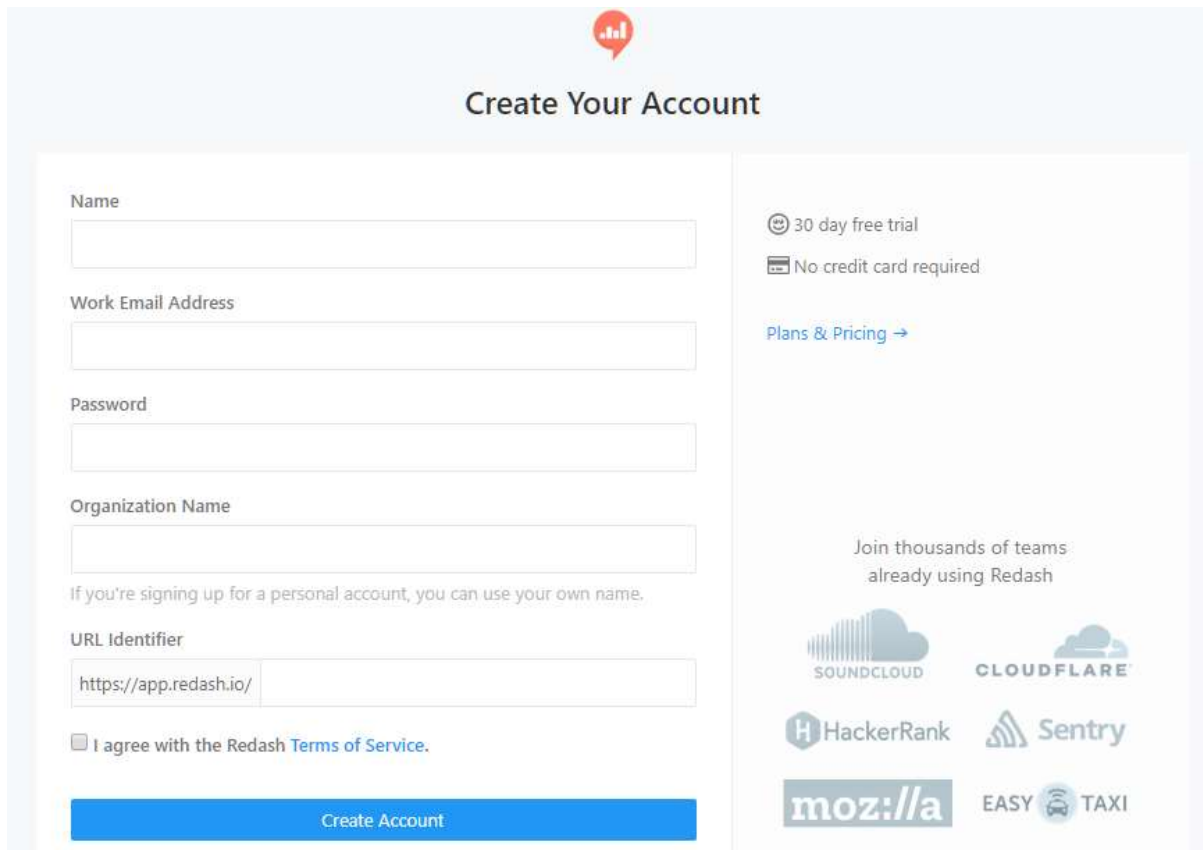
Download .csv

	User	Access key ID	Secret access key	Password	Email login instructions
▶	✓ EE5111_A0...	AKIASPTUJXSK7MD26GP2	***** Show	***** Show	Send email ↗

Figure 45: AWS IAM Table for Things part 6

Redash an open source tool used for Thing1 and Thing2 in this assignment by connecting and querying data sources from AWS DynamoDB, it can also build dashboards to visualize data and share them with the public through URLs. Redash is quick to setup and works with any data source to begin, sign up with Redash and 'Create Your Account' as shown in Figure 45.

Redash data visualization



The image shows the 'Create Your Account' page on Redash. It features a form with the following fields: Name, Work Email Address, Password, Organization Name, and URL Identifier (pre-filled with 'https://app.redash.io/'). A checkbox for 'I agree with the Redash Terms of Service' is located below the URL field. A blue 'Create Account' button is at the bottom of the form. To the right of the form, there is promotional text: '30 day free trial', 'No credit card required', and a link to 'Plans & Pricing'. Below this, it says 'Join thousands of teams already using Redash' and lists logos for SoundCloud, Cloudflare, HackerRank, Sentry, moz://a, and EASY TAXI.

Figure 46: Redash data visualization part 1

After creating an account, a new query can be created at Redash main page as shown in Figure 46.

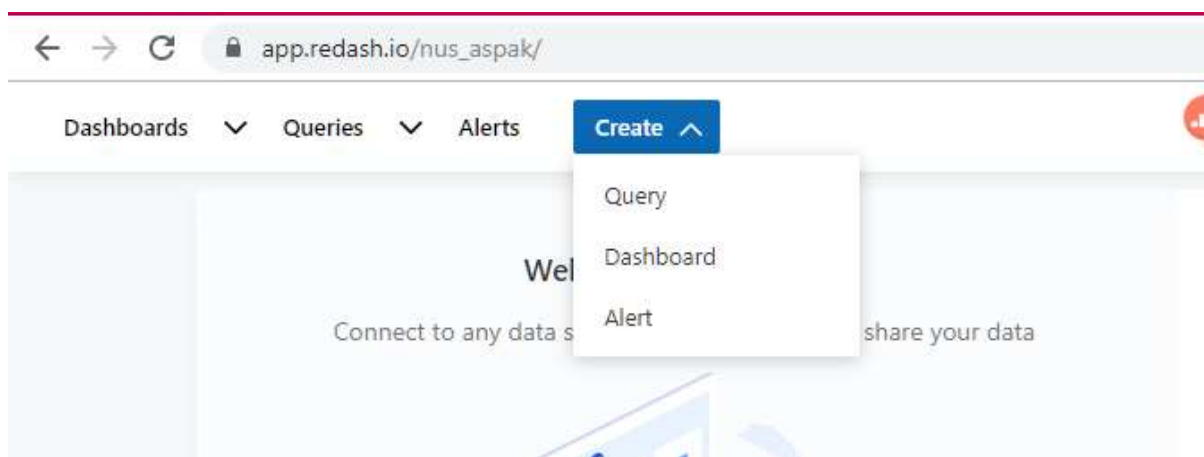


Figure 47: Redash data visualization part 2

To link Redash query to AWS DynamoDB table, go to 'Data Sources' and select 'New Data Source' under settings as shown in Figure 47.

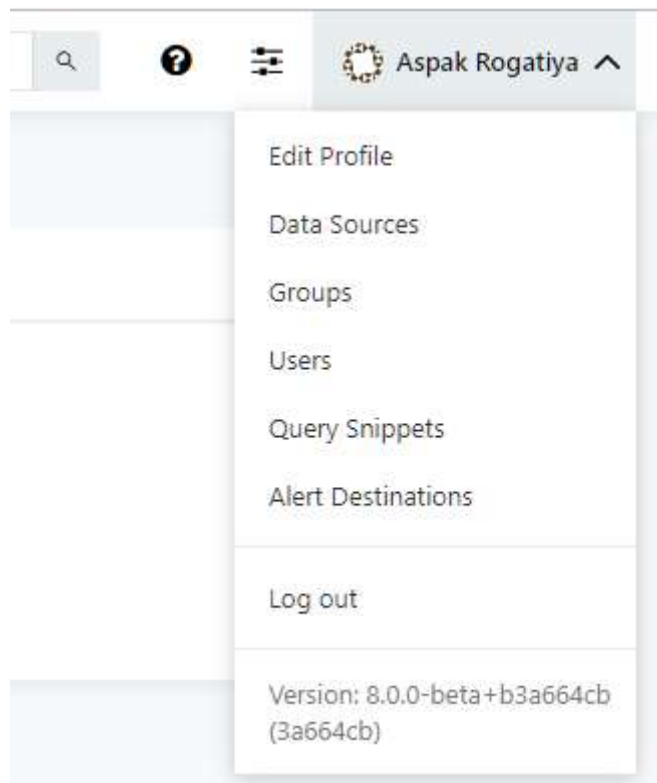


Figure 48: Redash data visualization part 3

Under 'Create a New Data Source' search for 'DynamoDB (with DQL)' and select 'Create' as shown in Figure 48.

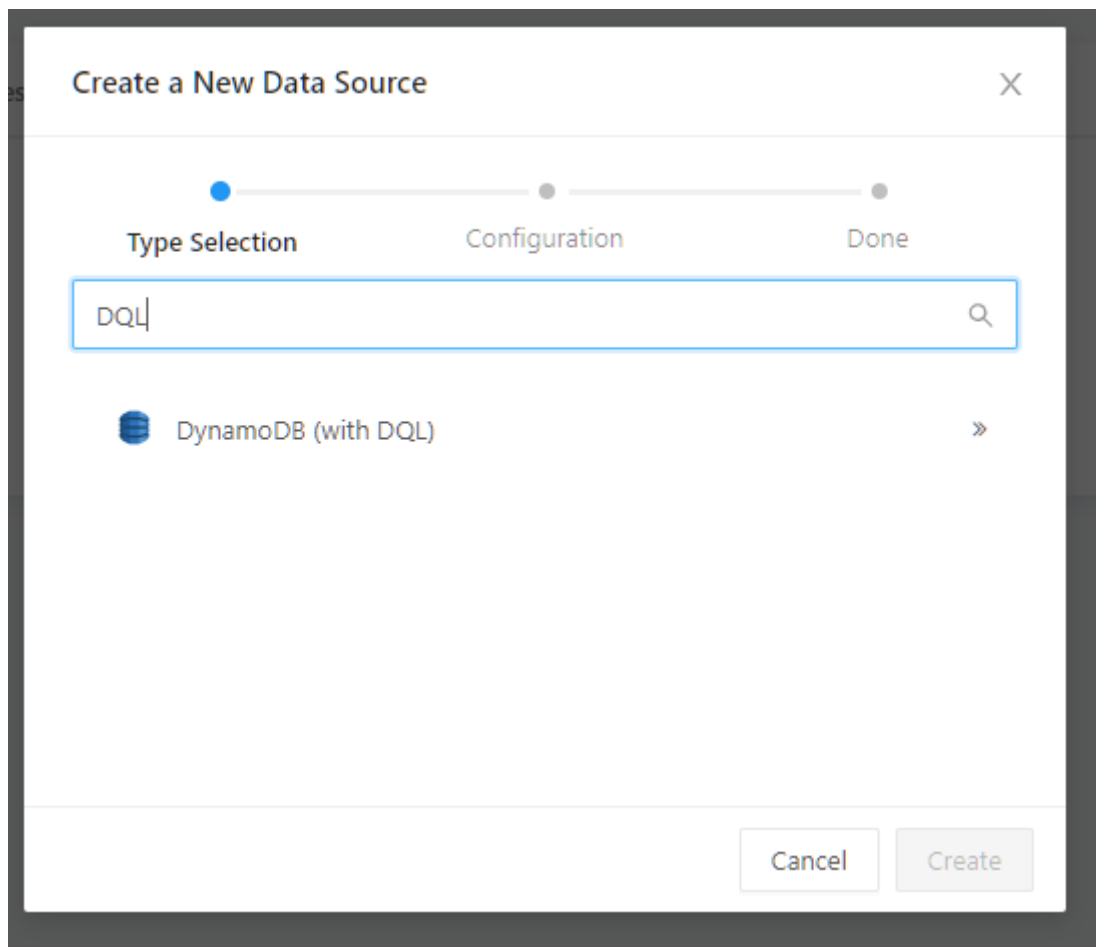


Figure 49: Redash data visualization part 4

Enter all 4 rows namely 'Name', 'Access Key', 'Region' and 'Secret Key' which can be obtained from 'credential' csv downloaded earlier and select 'Create' as shown in Figure 49 and 50.

The screenshot shows the Redash 'Data Sources' configuration page for a new AWS DynamoDB connection. The navigation bar at the top includes links for Data Sources, Users, Groups, Alert Destinations, Query Snippets, Settings, Account, and Plan & Billing. The main heading is 'DynamoDB (with DQL)' with a database icon. The configuration form contains the following fields:

- * Name:** EE5111_A0179741U (Validated with a green checkmark)
- * Access Key:** AKIASPTUJXSK7MD26GP2 (Validated with a green checkmark)
- Region:** us-east-2 (Validated with a green checkmark)
- * Secret Key:** (Masked with dashes, validated with a green checkmark)

At the bottom of the form, there is a large blue 'Save' button, a grey 'Delete' button, and a 'Test Connection' button. A success toast notification is visible in the bottom right corner, displaying a green checkmark, the word 'Success', and a close button (X).

Figure 50: Redash data visualization part 5

In Redash query interface, input the query 'Name', enter the SQL query codes for all the sensors from jet engine, 'Save' and select 'Execute' as shown in Figure 51. Loading of the data from AWS DynamoDB to Redash maybe take some while depending on the amount of data to be transferred. SQL query codes can be found in the appendix.

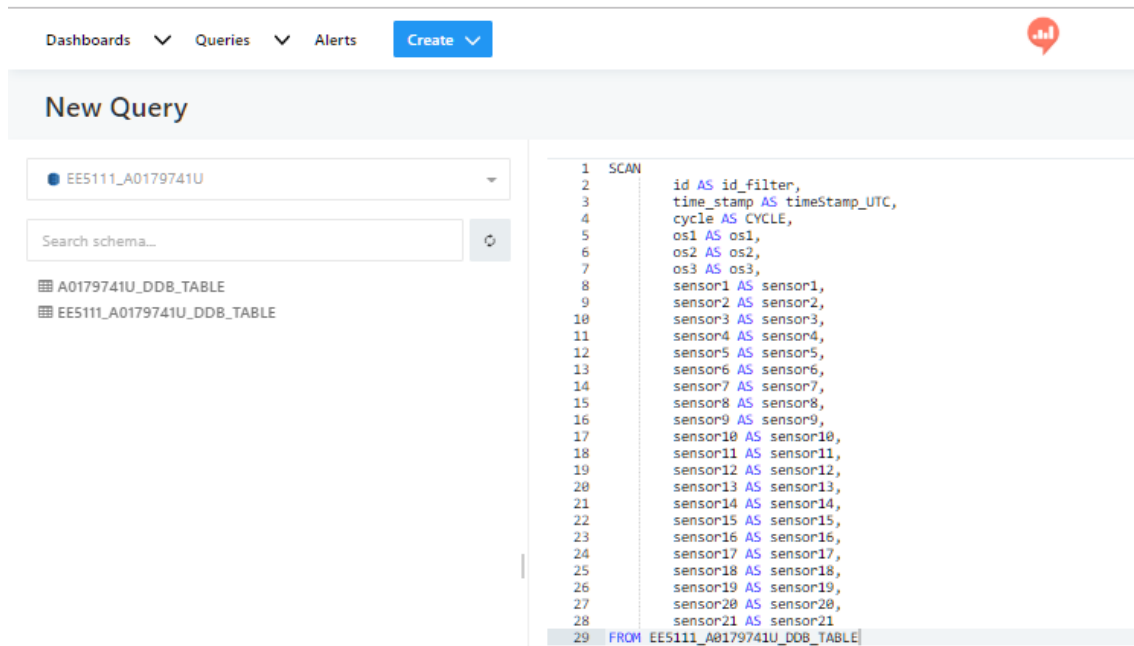


Figure 51: Redash data visualization part 7

Once all data have been export over, it will be displayed on the interface. Select 'New Visualization' to create data visualization for the Things as shown in Figure 52.

id_filter	timestamp_utc	cycle	os1	os2	os3	sensor1	sensor2	sensor3	sensor4	sensor5	sensor6	sensor7	sensor8	sensor9	sensor10
FD002_1	1.5690861071.38	1	54.8883	0.8400	100.0	449.66	955.23	1188.91	1127.23	3.48	8.00	194.64	2222.95	8341.31	1.02
FD002_1	1.5690861083.99	2	41.8982	0.8409	100.0	441.00	949.60	1193.22	1125.79	3.81	5.71	138.51	2211.57	8305.96	1.02
FD002_1	1.5690861093.99	3	24.9988	0.8219	90.0	462.54	927.31	1236.76	1047.48	7.05	8.00	175.71	1915.11	8001.42	0.94
FD002_1	1.5690861104.01	4	42.0077	0.8416	100.0	449.88	949.31	1194.03	1126.38	3.81	5.71	138.48	2211.58	8305.96	1.02
FD002_1	1.5690861114.04	5	29.0000	0.8103	90.0	482.56	937.07	1237.71	1047.89	7.06	8.00	175.05	1915.10	7985.23	0.94
FD002_1	1.5690861124.06	6	29.0045	0.8105	90.0	482.56	937.03	1236.34	1048.70	7.06	8.00	175.17	1915.15	7986.10	0.94
FD002_1	1.5690861134.08	7	42.0040	0.8409	100.0	449.88	949.74	1194.45	1127.16	3.81	5.71	138.71	2211.62	8307.75	1.02

Figure 52: Redash data visualization part 8

Under visualization editor, enter a 'Visualization Name' for the Thing1, choose chart type to be 'Line' graph, X column to be 'Timestamp_utc', Y column to be 'sensorX', id to be the 'cycle' of interest and select 'Save' as shown in Figure 53.

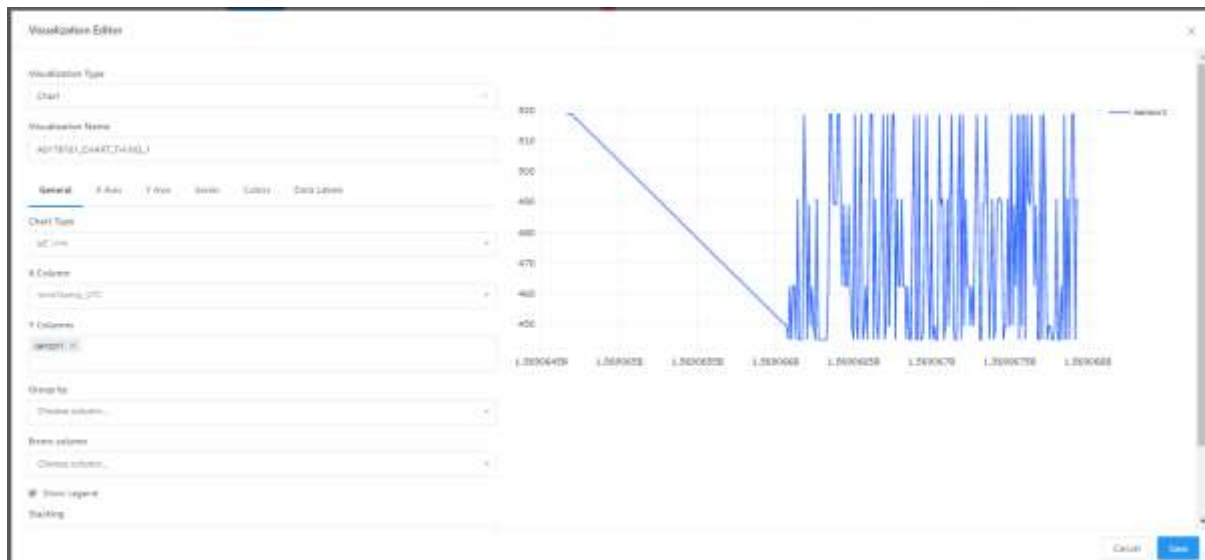


Figure 53: Redash data visualization part 9

Repeat the same procedure above for Thing2 for both Things to be visualized in dashboard later as shown in Figure 54.



Figure 54: Redash data visualization part 10

Redash dashboard allows combine visualizations and text boxes that provide context with the data. On Redash interface, select 'Create' new dashboard and enter a 'Name' for the dashboard as shown in Figure 55.

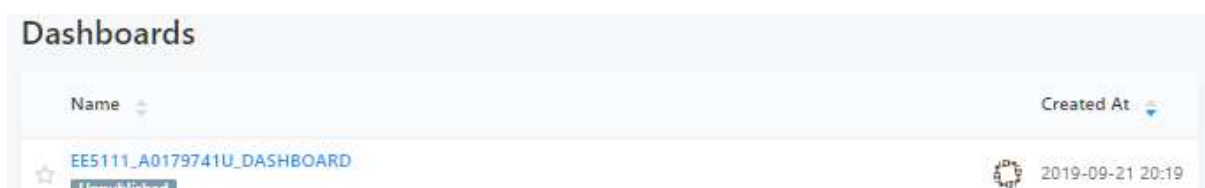


Figure 55: Redash data visualization part 11

After creating the dashboard, go back to the query created and select 'Add to Dashboard' as shown in Figure 56.



Figure 56: Redash data visualization part 12

Under 'Add to Dashboard', select the dashboard name created earlier. Repeat the same procedure for both Thing1 and Thing2 as shown in Figure 57.

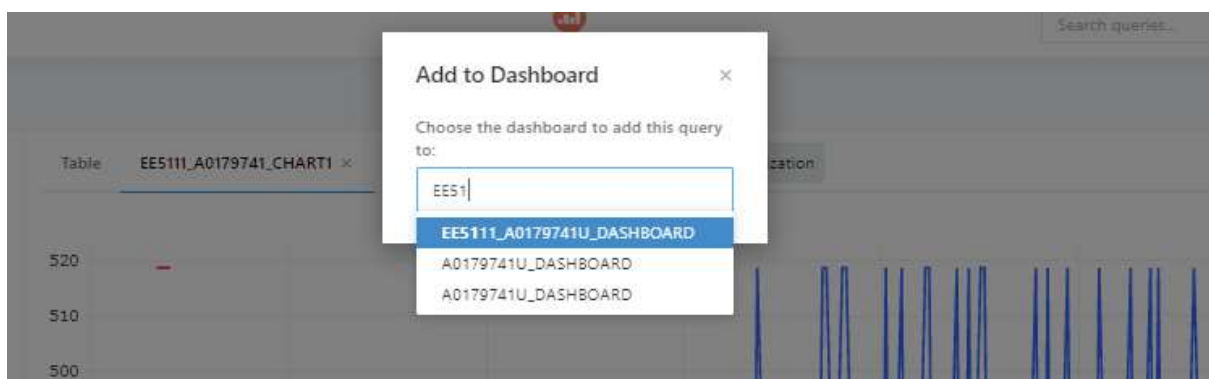


Figure 57: Redash data visualization part 13

On Redash interface, choose Dashboard to view the queries chosen to be displayed. More sensors data can be added in its query interface or different cycle of the things can be selected by changing the 'id'. Public ULR of the dashboard can also be set public as clicking the 'Share' icon as shown in Figure 58.

https://app.redash.io/nus_aspak/public/dashboards/qJA2vf1INzZ2kJVFUG4XwskXBfH1aUmRN3SNWkDW



Figure 58: Redash Dashboard